DESIGN AND IMPLEMENTATION OF AN ENGINEERING DRAFTING GRAPHICS PACKAGE

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

by

C. H. VENKATESH MURTHY

to the

COMPUTER SCIENCE PROGRAM
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

JULY, 1982

CERTIFICATE

This is to certify that the thesis entitled 'DESIGN AND IMPLEMENTATION OF AN ENGINEERING DRAFTING GRAPHICS PACKAGE' has been carried out by Shri C.H. Venkatesh Murthy under our supervision and has not been submitted elsewhere for the award of a degree.

Dhande

Dr. S.G. Dhande
Assistant Professor
Department of Mech. Engg.
and
Computer Science Program
I.I.T. Kanpur.

R Sankar

Dr. R. Sankar
Professor
Computer Science Program
I.I.T. Kanpur.

ACKNOWLEDGEMENTS

I am grateful to Dr. R. Sankar and Dr. S.G. Dhande, my thesis supervisors, for their constant help and guidance throughout this project.

I am grateful to my service, the Defence Research and Development Organisation (DRDO), for having provided me the opportunity to work towards my M.Tech. degree at I.I.T. Kanpur.

I would like to express my hearty thanks to Lt.Ramaprasad for having assisted in preparing the drawings.

I would also like to thank all my colleagues and friends for their help at various stages of the work.

I must thank Mr. C.M. Abraham for his excellent typing.

C.H. Venkatesh Murthy

ABSTRACT

A software package for computer-aided drafting of engineering drawings has been designed and implemented. Using a set of commands and controlling the position of the cursor through a set of keys of a graphics terminal, the user is able to draw an engineering drawing. A simple and compact data structure has been used to represent and store graphic entities. It is possible to edit the drawing and to define a drawing in a level-wise manner. Macro facility has been provided for posting several instances of a segment in a drawing. A filing facility has also been provided for storing the data and for retrieving it for later modification and documentation. The software has been developed in the PASCAL language and has been implemented on the DEC-1090 system using a Tektronix 4006-1 terminal. Some illustrative examples showing how the package can be used are given in the present work.

LIST OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 COMPUTER-AIDED DRAFTING

In order to convey ideas about objects to be designed and manufactured, engineers generally employ the means of drawings showing the projected views of the object. The projections used are : multiview orthographic projections, axonometric projections, oblique projections or perspective projections. No matter what projection scheme is selected, it is necessary that complete information about the object to be described be given in the drawing in a clear, compact as well as in an unambiguous way. Since an engineering drawing forms a document that is used as a reference by a variety of departments in the manufacturing world, it has become customary to follow a set of standard codes for generating these drawings. For instance, a hidden line is shown as a broken line with the length of the dash as 3 mm and the gap between successive dashes as 3 mm.

Management of information is a major problem of the present technological world. However, using computers it has been found that this problem can be brought under control. Similarly, managing graphics information is also, a major problem in engineering organizations. A large number of drawings are generated and stored in the form of

archives. The process of retrieval, updating and editing is very laborius and time consuming. It has now been felt that using computer graphics hardware and software the problem of managing graphics information can be overcome. In short, it is necessary to develop the appropriate software compatible with the available computing facilities and the computer graphics hardware. The software should confirm with the standard drafting practices or codes and should provide enough facilities of management in the form of editing, filing and retrieval. The present work is an attempt to develop such a drafting package.

1.2 GRAPHICS HARDWARE

In general, a draftsman uses a drawing board or a drafting machine, a set of drawing instruments, a set of pens or pencils and some other accessories like scales, eraser etc. In a typical system of computer graphics, the commonly used tools are : a graphics terminal with a keyboard, a digitizer or a tablet, a plotter, a computer and the necessary software. With this kind of a system, one can prepare drawings from rough sketches to a final form without handling the conventional drafting tools. Details regarding the features of the computer graphics equipment have been given by Scott [5], Newman and Sproull [2] and Machover [6]. Details regarding some of the graphics software and its relevance to the drafting work have been given by Besant [1]

and Scott [5].

1.3 SCOPE OF THE PRESENT WORK

In order to develop a general purpose drafting package, the following considerations have been taken into account.

1. It is presumed that the user is not an experienced programmer. Hence, the commands of the software are kept to a small number and the interactive features are simple in nature.

2. The command language is not elaborate with the hope that not much time will be required for user training. However, enough provisions have been made to detect and to point out an error committed by a user.

3. The entire package has been implemented in a higher-level language PASCAL without using any general-purpose graphics package such as GPGS, PLOT10 etc. This, it is hoped, will enhance the portability of the drafting package.

The specifications of the package alongwith its design considerations are given in Chapter 2. The details regarding the implementation of the graphics package are given in Chapter 3. The geometrical considerations that are used in several procedures such as searching a line or generating a circle are given in Chapter 4. Several illustrative examples showing how the facilities provided in the package can be used are given in Chapter 5. Also given in Chapter 5 is a list of suggestions for further work.

CHAPTER 2

SPECIFICATION AND DESIGN OF THE SYSTEM

2.1 SPECIFICATIONS

2.1.1 Primitives

An engineering drawing, generally consists of different types of lines, circular arcs and circles. The types of lines that are used most often are :

a) Solid lines (of different thickness),

b) Dotted lines,

c) Central or chain lines, and

d) Dimension lines consisting of

   i) leader lines (arrow at end)

   ii) leader lines (arrow at start)

   iii) leader lines (arrow at both ends)

The types of arcs that are encountered are same as the lines mentioned above.

The types of circles, which are most commonly found are :

a) Solid circles,

b) Dotted circles,

c) Chain circles.

It is desirable to provide the above types of lines, arcs and circles as basic primitives in a drafting graphics packages.

2.1.2 Filing Facilities

The system should be centred around the drawing library on disc, which contains all the drawings created by a user. Different users should be able to create and maintain their own library individually. A user should be able to list the directory of his drawing library and should also be able to obtain a copy of any drawing he desires.

2.1.3 Editing Facilities

The main benefits of the drafting system come from rapid modification of existing drawings. A powerful editor can provide the user, a rapid manipulation and editing of drawings that have already been totally or partially created.

2.1.4 Levels

Levels are used to divide up drawing data, so that a complicated drawing can be broken up into a series of sub-drawings. One of the advantages of levels in drawing is that if the user finds some part of the drawing is totally wrong, then if he has inputted that part of the drawing under some level, he can easily delete that part by specifying the level.

2.1.5 Hatching

Very often, in mechanical drawings hatching is used to make sections evident. Therefore, it is very useful to have a facility for hatching,which accepts as input data, the area to be hatched. Here, it can be seen that levels are useful in specifying the area to be hatched.

2.1.6 Macros

Macros are standard components or items that are in regular use. A macro facility greatly enhances the drafting system in that similar items, components, or sections of data may be created only once, filed, and subsequently reproduced with little effort.

The user must study a problem in advance and decide if groups of data can be treated as macros, since it is repetition which can be exploited in drafting.

2.1.7 Command Language

The success of any interactive system seems to be closely connected to the interface between the system and the user. A well adapted command language plays an important role in this connection.

A command language may be looked upon as a communication bridge between the user and the system. Therefore, it is important that the design and implementation of the language is adapted to the user's needs in order to enable him to

to utilize his knowledge and ideas without getting disturbed by a dialogue with the system.

2.1.8 Windowing and Clipping

When it is necessary to examine in detail, a part of the drawing being displayed, a window may be placed around the desired part and the windowed area magnified to fill the whole screen. This involves scaling the data which lies within the window so that the window fills the entire screen. Data that lies outside the window must be eliminated so that only the data required for display is processed. This process is known as clipping.

Some hardware devices have automatic scissoring in which the window and the display vectors may be larger than the display size. For some systems, as with the present DVST system, hardware clipping is not available and software clipping must therefore be employed.

It is necessary to define two viewing areas, these being a viewport and a window. It is useful to make the viewport equal in size to the screen to take advantage of maximum screen area. Normally, the screen represents the whole drawing area. The maximum size of the drawing area is usually equal to the plotter size.

2.1.9 Text

Text strings of different sizes and shapes and at various angles are required to annotate the drawing. Most of the display devices and plotters will have the capability of generating text strings of different sizes by hardware.

2.2 DATA STRUCTURE AND STORAGE

One of the first decisions made in designing the drafting system specified in Section 2.1 concerned the type of structure and format to be used for the storage of drawing data.

A number of data structures have been proposed and implemented in interactive graphics systems to provide powerful facilities for design/drafting [3,4]. Depending upon the application requirements, varying degrees of sophistication and complexity are employed in data structures, resulting in systems with different degrees of interactive capabilities.

The data structure used most generally in large graphics systems is the hierarchical ring structure because it offers the greatest flexibility. Flexibility, however, also poses the biggest problems. The more flexible a structure is, the more difficult it is to implement. Also, the more flexible a structure is, the more pointers it uses and therefore, the more memory space it requires.

In many non-engineering applications, a drawing is composed of a series of repeated graphic elements. Typical examples are electrical or electronic circuits, or plant drawings. The data structure of such a drawing may be organized by establishing basic patterns used as primitives and more complex figures, built up by means of such primitives, which may in turn be appropriately repeated in generating the final drawings.

In the case of mechanical engineering drawings, it is not possible to identify any basic primitives which may then be suitably introduced into a hierarchical structure and hence the drawing will be structured at the lower graphic level of points, line segments and arcs [3].

The data structure, which represents an engineering drawing must contain all the connections among the graphic elements of which the drawing is composed. Also, while designing a data structure one has to have a compromise between storage and computation. What is the best compromise solution for a particular application? It has been felt that the prevalent practice is to over-pointerise a data structure in the belief that the savings made in computation and response times are justified by the extra storage used. However, we propose a data structure which uses minimum storage and which is simple to manipulate either in main memory or directly on disk.

The proposed data structure for the system specified in Section 2.1 is a simple linear list type structure as shown in Fig. 2.1(a).

A line can be stored as an element of a list containing the following information (Fig. 2.1(b)).

a) OPcode - indicates type of line

b) Level

c) $x_0 y_0$ - starting point

d) $x_1 y_1$ - ending point

But the above configuration seems to be storagewise inefficient. Because most of the times the lines are drawn contiguously. That is ending point of one line will be the starting point of another. Therefore, there can be a high redundancy in the stored points.

An alternative approach suggests that let each record of a list contain only one point with an additional information about that point, that is, whether the point is a beginning point of a line. One bit of information will do for this. The bit will be zero if it is a beginning point of a line and one otherwise.

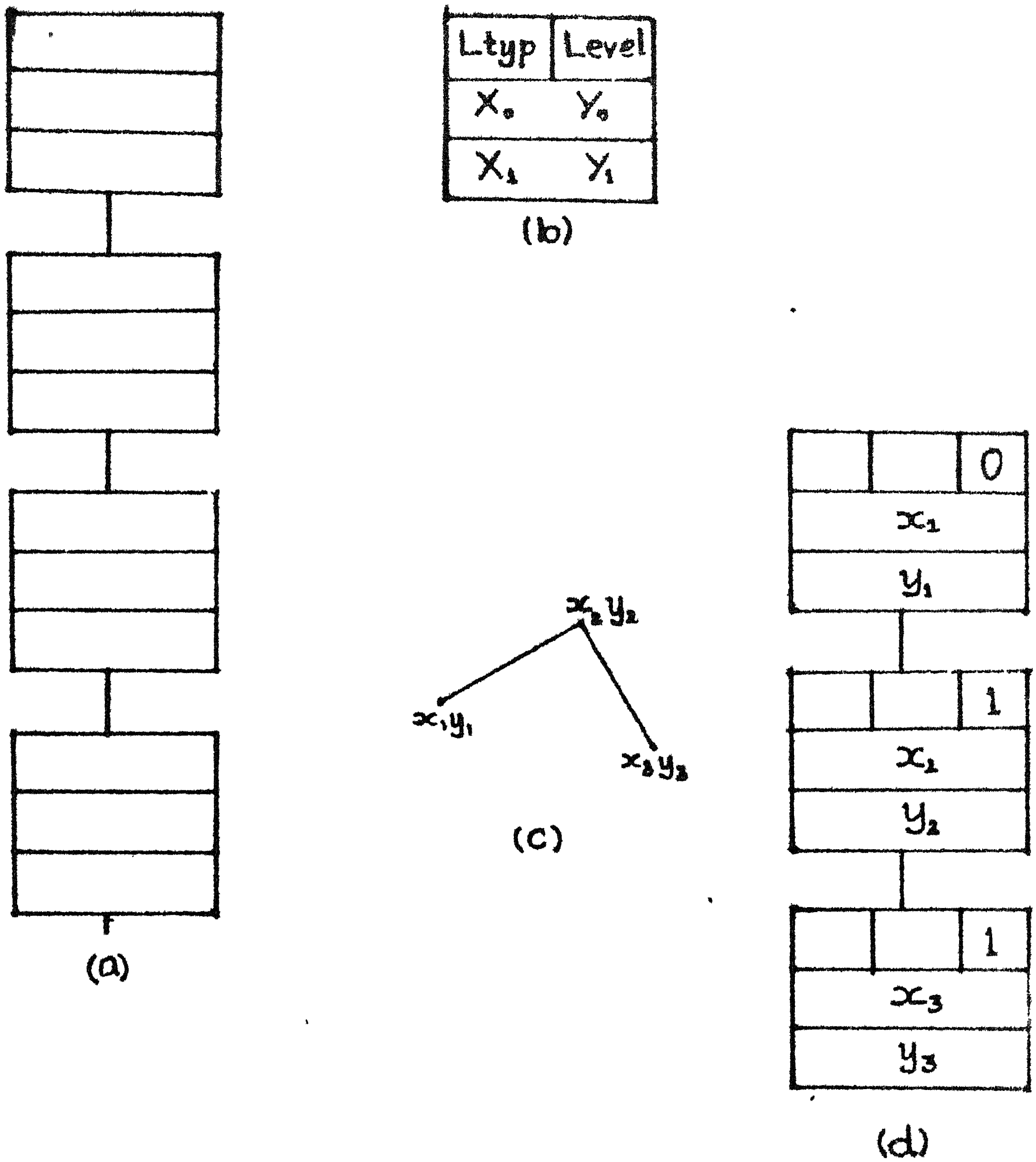


Fig 2·1 (a) general data structure as a linear list. (b) Inefficient way of representing a line. (c) line data. (d) representation of line data of fig. (c).

Now the record in a list will contain the following information for lines.

a) OPCODE

b) Level

c) Status

d) x, y

Now the representation of data structure for Fig. 2.1(c) looks like Fig. 2.1(b). For editing purposes one more bit of information for each record is required to indicate that the record has been deleted. Let us call this bit as D.

Now to decide about the size of the record, we may have to consider the possibility of packing certain information into a single word. If we decide to have maximum number of OPCODE associated with a point to be 64 and the maximum number of levels to be 32. We can pack OPCODE, Level, Status and Delete information into one word of any computer system which is having word length of atleast 16 bits.

The x and y coordinates occupy each one word. Therefore, the size of the record in a list is fixed to 3 words.

Now, let us call all graphic items which are not straight lines as symbols. Therefore, arcs and circles are now being considered as symbols.

Symbols are defined as being program-generated sub-pictures; that is, the symbol data is used to call a section of program to generate the specified symbol picture. Therefore, symbols are generally stored in the data structure with the minimum amount of data necessary for the system to define them completely. All symbols follow these convetions.

Now, the circle can be stored as a centre point in one record and its radius in the second record. Two more records are used to indicate high and low limits on the symbol respectively (Fig. 2.2(a)). This is called the bounding box information. This information is useful to the symbol editor and to the display routines to decide whether the symbol is within the current area of interest. In short, a total of 4 records are used to represent a circle. In a similar way an arc can be represented by using five records as shown in Fig. 2.2(b).

Strings can also be stored, as symbols with the starting point x_s, y_s in the first record and in the subsequent records string data is packed as tightly as possible (Fig. 2.2(c)). The end of the string is indicated by a special marker.

Macros can also be treated in a similar manner as symbols and we can economise data storage by storing macros only once and referencing it by a code in the data

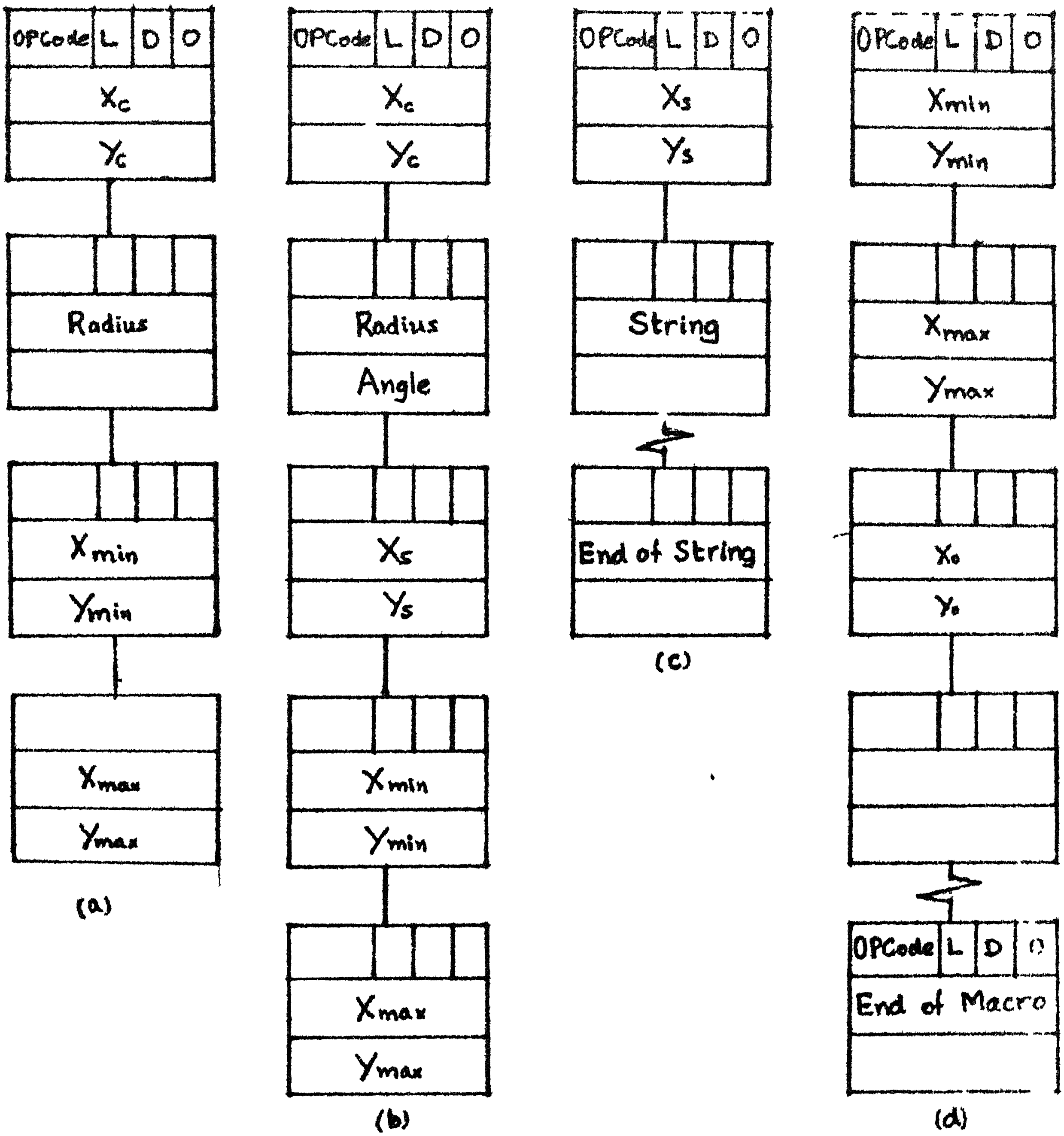


Fig 2.2 (a) Representation of a circle in the data structure. (b) Representation of an arc. (c) Representation of a string. (d) Representation of a macro.

list specifying not only the macro used but its scale, rotation, position, etc.

The disadvantages of this system of data structure for macros are as follows :

1) Editing the contents of a macro when it has been positioned on a drawing becomes difficult.

2) Display time is slightly increased as a transformation has to be applied to the macro before it is displayed.

3) Finding a point or line within a macro requires a complicated search routine.

For these reasons it was decided that in the present drafting system macros should not be stored separately, although it is recognised that applications where many standard items are used , could use such a system to great advantage, for example, electronic circuit design.

Fig. 2.2(d) shows the structure for storing macros. The beginning two records contain low and high limits of the macro respectively. The third record contains the origin. The last record in a macro is a dummy record and indicates the end of macro.

2.3 ORGANIZATION OF THE DRAFTING GRAPHICS PACKAGE

Fig. 2.3 shows the block diagram of the organisation of the graphic package specified in Section 2.1. The

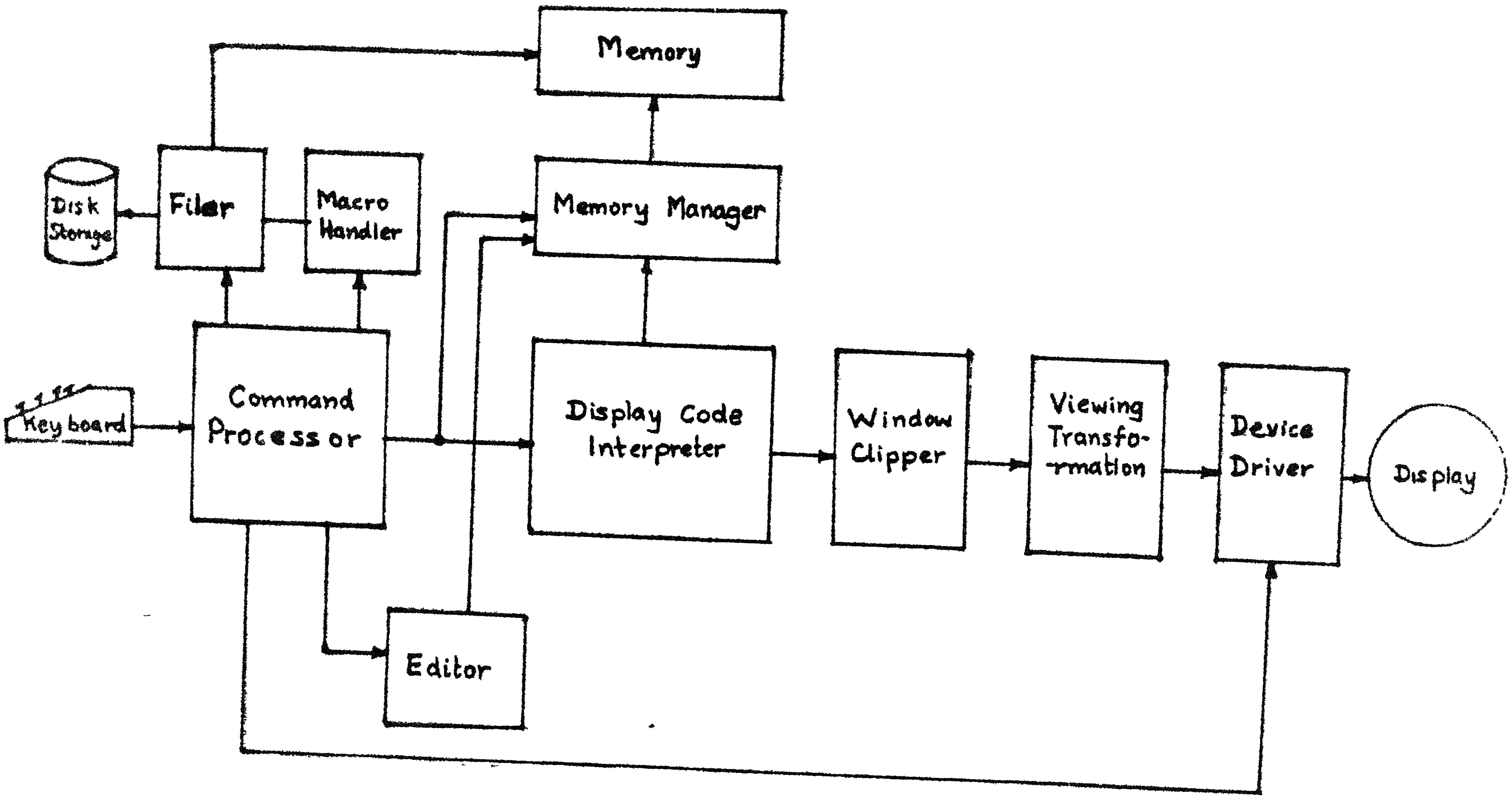


Fig 2.3 Organisation of graphics package.

general description of the function of each module is given below. Implementation details of each module are described in Chapter 3.

2.3.1 Device Driver

In this package this module is written for Tektronix 4006-1 storage tube display terminal. This terminal accepts point and vector display information in a certain fixed format. The basic function of this module is to convert the point and vector information into an appropriate command format.

2.3.2 Viewing Transformation

This module performs the picture transformation from user coordinates to screen coordinates.

2.3.3 Window Clipping

It is desirable to clip the drawing data at the user coordinate level itself than clipping it after transforming to screen coordinates. This eliminates lot of unnecessary computation which is performed to the data that lies outside the window. Therefore, to eliminate all the data that lie outside the user window, clipping is performed at window level. This module performs the clipping at window level.

2.3.4 Command Processor

This is one of the important modules in the graphics package. This forms the vehicle between the user and the

system. This module responds to all the commands given by the user and checks for validity of commands, and reports to the user if there are any errors in the command. On receiving valid commands, depending upon the type of command, it takes appropriate actions. For example, if it is an editing command, then the editor is invoked else if it is a line drawing command then appropriate code is generated and then the display code interpreter is called to interpret the code.

2.3.5 Display Code Interpreter

This module scans through the data structure, interprets the display code and generates the appropriate display. This module basically consists of a set of procedures. These procedures generate the basic primitives that are provided in the package (see Section 2.1.1). The main procedure in the display code interpreter calls one of these procedures by passing appropriate parameters to it. The type of procedure called depends upon the type of display code encountered.

2.3.6 Editor

There are four types of editors : Line editor, Symbol editor, Macro editor and Level editor.

2.3.6.1 Line editor

This deletes the line indicated by the user in a

drawing. To indicate a specific line in a drawing, the user must be provided with a facility to point to a line, which he desires to delete without any ambiguity.

In the case of a line, we have two defining points. Further more, each end point of a segment may also be an end point of another segment, if the segments form a polygon. To identify a line, in a simple scheme of comparing the cursor coordinates with a defining point of the object requires that only one of the two end points be used consistently with comparison. Actually, many systems operate that way, requiring the user to position the cursor, for example, near the tail of the vector that is to be picked. Such an approach has certain short comings. For example, consider a case in which each end point is an end point of more than one edge. By which rules shall a line segment be uniquely identified? Moreover, how is the user to know what is head and what is tail? These problems can be avoided by measuring the normal distance between the cursor positions and a line. Thus, by placing the cursor anywhere along the line within certain distance, the system must be able to search and identify the line to be deleted. This line editor routine search for such lines and if found any, flashes the cursor between end points of a line searched. If the user does not wish to delete this particular line, he can ask the system to search for a new line.

2.3.6.2 Symbol editor

As mentioned earlier in Section 2.2, all graphic items which are not straight lines are treated as symbols. It is important to be able to identify such symbols and to delete these as one unit. This routine makes use of the high and low limit information stored in the data structure for each symbol. mLike the line editor, the user must place the cursor so that it lies within the low and high limits of the symbol. This routine checks the limit of each symbol until the appopriate one is located.

2.3.6.3 Macro editor

This is almost similar to symbol editor. The user is required to place the cursor anywhere within the high and low limits of the macro. The routine searches for the appropriate macro and it can be deleted as a whole.

2.3.6.4 Level editor

The user may choose to delete all data in a particular level. This is useful if the user has planned his drawing correctly and has inputted sections in different levels. If one section turns out to be completely wrong, this is a much faster method of deleting it than using the symbol and line editors to remove each data item individually.

2.3.7 Filer

This routine helps in storing the drawing information on

a secondary storage. The user can specify the file name in which the drawing has to be stored. Each drawing is associated with a name. The filer maintains a directory of drawings for each file. The filer can be asked to list the directory. It can also load the drawing information which is stored on the disk into the user work space for editing purposes. This also retrieves the drawing specified by the user for outputting on the plotter.

2.3.8 Macro Handler

This routine enters into the users work space, the specified macro from the macro library. When the user recalls the macro by name, the macro handler requests the filer to retrieve the macro and then it applies appropriate transformation specified by the user to the macro and loads into the workspace.

2.3.9 Memory Manager

This manages the memory used for storing the data structure. It allocates on request from the command processor, the available storage to maintain the data struture structure for drawings. When it runs out of storage space, automatic garbage collection is done to retrieve the space occupied by the items which have been deleted by the editor.

CHAPTER 3

IMPLEMENTATION OF THE SYSTEM

In this chapter we shall discuss in detail the implementation of the drafting package described in Chapter 2.

As mentioned in Chapter 1, one of our objectives has been to make the package portable. In other words, the package should be machine independent. Machine independence in graphics package design can be achieved by implementing the package in a widely used language, such as PASCAL, FORTRAN, ALGOL or APL. In our case, it was decided to implement the package in the language PASCAL. PASCAL can be considered as a very desirable environment for an implementation of this package because, PASCAL is a block structured language that combines the concepts of outstanding clarity and simplicity with a modern control structure and a variety of data types. Moreover, the language PASCAL is getting more and more popular and is available on most of the recent computer systems.

3.1 THE COMMAND LANGUAGE

The design of the command language is directly related to the type of input device used in the interactive design. Our design is based on the alphanumeric keyboard as the input device, wherein, the user interacts with the system by typing a set of commands.

The first problem in designing a set of commands is that of language design itself. One must define a syntax; must make sure that this syntax is consistant; must determine what syntax errors the user can make; and must make sure that the syntax is sufficiently comprehensive and powerful to satisfy the user's needs. The language design involves consideration not only of syntax but also of semantics, i.e., the meaning to be attached to each syntactic construct. The last issue to be considered, while designing a command language is that how the program should respond to meaningless or erroneous inputs. In many cases errors should simply abort the current command and inform the user of his error.

Based on the above discussion, a keyboard command language has been designed. The syntax and semantics of the language is discussed in detail in Appendix A.

To make use of the package effectively, the user must learn the syntax and semantics of the command language. In order to shorten the user's learning period and simplify implementation, the language has been designed with a very simple syntax. It basically consists of the action verb specifying what action is to be taken followed by the operands, if any, to which the verb applies. For example, the user might type

LS L:10 A:45

in order to draw a solid line of length 10 units at an angle of 45^{o} from the present cursor position. From the above example one can illustrate the basic elements found in a command :

1) The verb (LS) specifying what action is to be taken,
2) The operands (L:10 A:45) to which the verb applies;
3) Delimiters (spaces, colon, terminating RETURN) between the elements.

Please note that these items in some cases can be omitted with the exception of the verb, which must always be present to indicate the action. For example, the user may just type

LS

omitting the operands. Under such circumstances, the program assumes default operands. In this case a solid line is drawn between the previously fixed cursor position to the present cursor position.

3.2 DEVICE DEPENDENT FEATURES

Most of the device-dependent parts of the package are those that contend with the functional characteristics and data formats of the output device. Therefore, to achieve device independence in a package, these components, which are inherently device-dependent have to be carefully separated from the remaining common software.

Two graphical primitive procedures have been written for the tTektronix 4006-1 DVST graphics terminal. The procedure MOVETO (P) (see the program listing in Appendix B) moves current beam position to P, where P is a point specified by its device screen coordinates. The second procedure DRAWLINE (P_0,P_1) draws a line from points P_0 to P_1. Again, both P_0 and P_1 are in screen coordinates.

The Tektronix 4006-1 DVST graphic terminal has a quality display area of 7.5x5.625 inches. The screen coordinate system is divided in to 1024 positions horizontally and 1024 positions vertically (780 of which are displayable). To address a point on the terminal, the x and y coordinates of the point are sent to the terminal in the form of four data bytes. Two bytes provide Y information and two bytes provide the X information. The bytes are sent to the terminal in the format HIY, LOY, HIX, LOX. See Ref.[7] for more details of the terminal.

The procedure SENDCOORDINATE (P) does the job of converting the x and y coordinates into the above mentioned format. To facilitate the user in seeing simultaneously, the graphic information and the command which he is typing, the whole screen is divided into two areas : namely, the graphic area and the command area, as shown in Fig. 3.1.

In addition to the above procedures two more procedures CLEARSCREEN and WORLDTOSCREEN (W,S) have been written. The

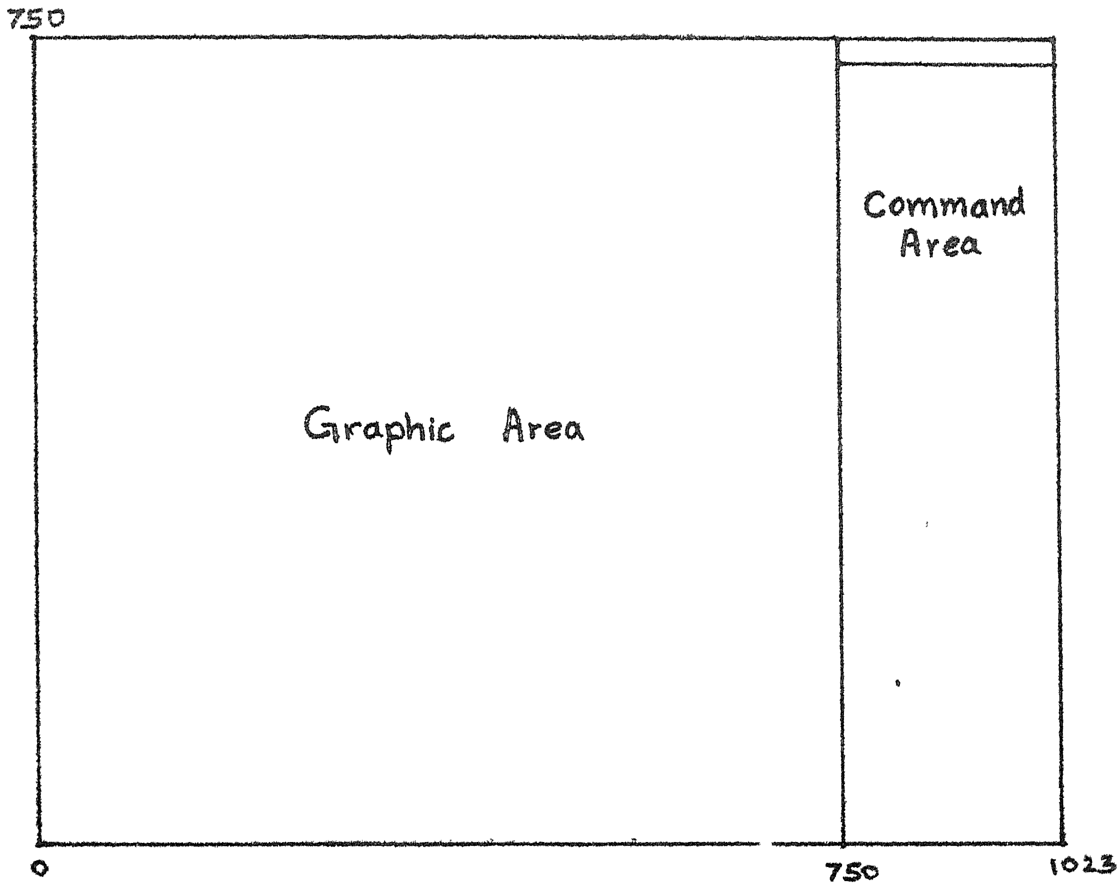


Fig 3·1 Division of screen into (a) graphic area. (b) command area.

procedure CLEARSCREEN clears the screen and the procedure WORLDTOSCREEN (W,S) transforms from user coordinate (world coordinates) to screen coordinates.

3.3 COMMAND PROCESSOR

The command processor has two modes of working, the monitor mode and the drafting mode. When the system is started, it will be in the monitor mode. In this mode, it responds to only monitor commands (see Appendix A). For example, the user can get various informations about his drawing and macro library, if any. He can list the names of drawings in the directory. He can get a copy of the drawing either on the terminal or plotter by calling the name of the drawing using TYPE command.

The mode is switched from monitor to drafting, when the user desires to create a new drawing or to edit the existing drawing. The functioning of the command processor in the drafting mode is depicted by means of a flow chart in Fig. 3.2.

There are two types of commands in this mode : single character or key commands and multicharacter commands.

3.3.1 Single Character Commands

It is necessary that for certain type of commands, the action is to be taken as soon as the key is pressed by the user. Some of the important commands and its functions are given below.

3.3.1.1 Cursor movement

Four alphanumeric keys have been assigned for the movement of cursor in the four directions. For the fine movement of the cursor, the same keys are used alongwith the SHIFT key. These keys are choosen in such a way that the character represented by these keys closely resembles the direction in which the cursor moves. For instance, the key with right arrow (closing angular bracket) is used to move the cursor to the right.

One can see that, assignment of keys like this has direct effect on the multiple character commands. Because, to distinguish between the single and the multiple character command, it is necessary that the first character of the multiple character command should not begin with the characters assigned to single character command. This problem could have been avoided had there been special function keys other than the ASCII character set available on the present Tektronix terminal keyboard.

3.3.1.2 Find command

This is another important single character command. This is more often used to find and position the cursor exactly on to the point already existing on the drawing. This can find either end points of a straight line or centre of an arc as well as a circle. This is useful in continuing the drawing exactly from a point already existing

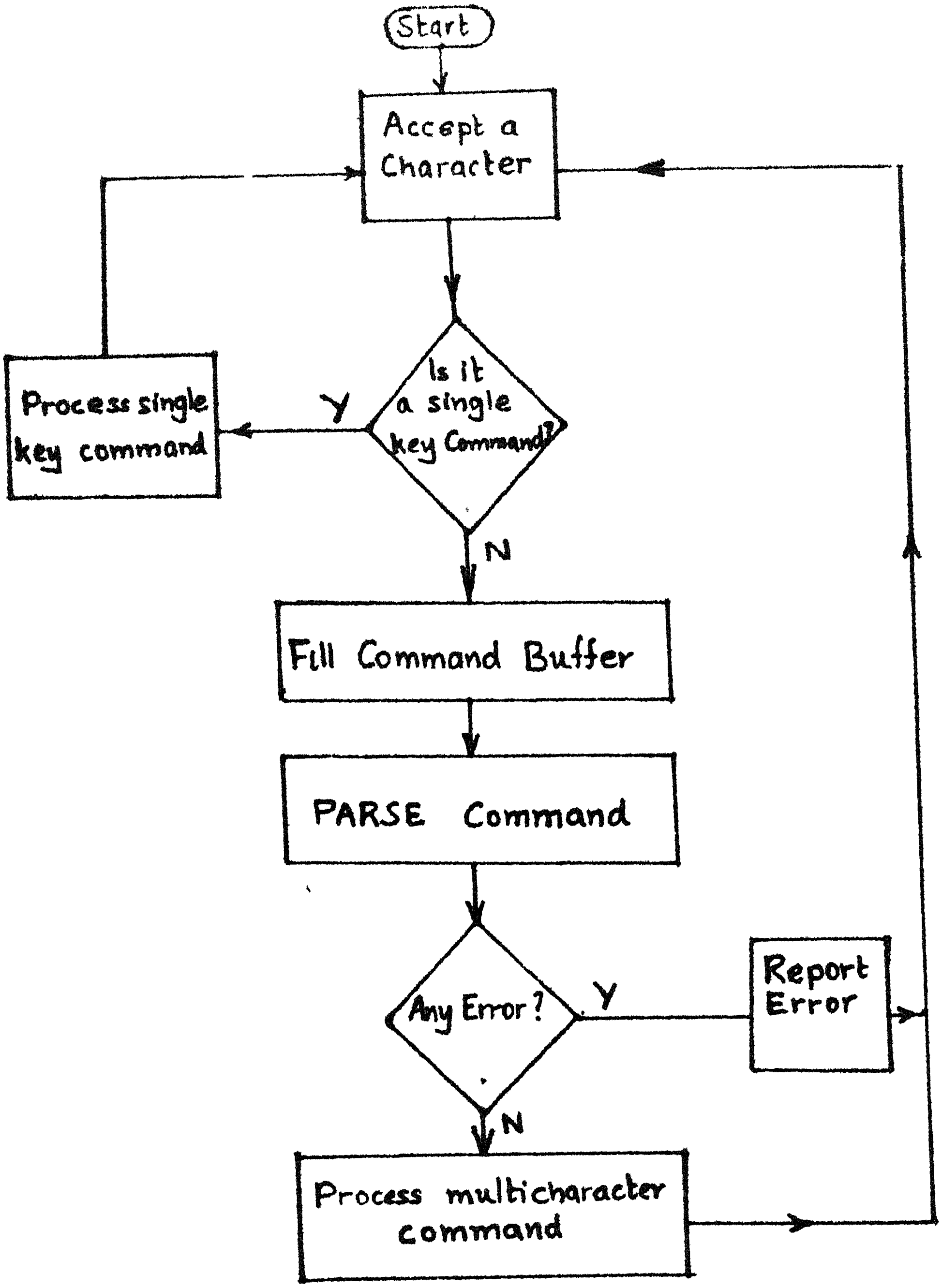


Fig 3·2 Flow chart of command processor in drafting mode.

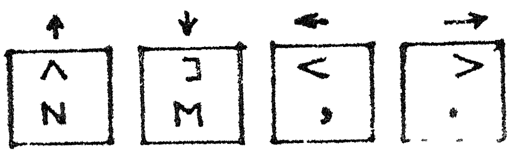


Fig 3·3 Keys for cursor movement. Arrow above the key indicates the direction of movement

on the drawing. This is also used to indicate precisely, the polygon to be cross-hatched using the hatching routine. To search a point, the user will have to position the cursor close to the point of interest. To search to a point near to the cursor, a simple scheme would be to define the 'fuzzinesh region' around the cursor. This region is defined as a square with the cursor in its centre as shown in Fig. 3.4. The FIND routine will search through the data structure for a point which lies within the fuzziness region. If the search is successful then the cursor is positioned exactly on to the point searched.

3.3.2 Multiple character Command

From the flow chart of Fig. 3.2, it can be seen that if the first character pressed by the user does not belong to the single character command group, the character is put in the command buffer. All the subsequent characters typed by the user are put in this buffer. This process continues till the command is terminated by the user on pressing carriage return. Then the command in the buffer is parsed. If there is any error in the syntax it is reported to the user and the whole command is cancelled. If there are no errors, depending upon the type of command, different routines are called to take the necessary action. For example, if it is a display generating command then the display code is generated and loaded into the data structure

and then the interpreter is called to interpret the generated code.

3.4 EDITORS

This routine is involved by the command processor on receiving the edit commands. Depending upon the type of editing, this routine in turn calls the appropriate routines.

3.4.1 Line Editor

This routine searches for a line in the data structure. When a line is located the cursor is flashed between the endpoints of a line and requests the user whether to delete this particular line or to search for a new line or to exit from searching.

3.4.1.1 How to search for a line?

It was mentioned in Chapter 2 that the user should be able to indicate the line to be deleted by placing the cursor anywhere along the line within the specified normal distance between the cursor position and a line (Fig. 3.5). Suppose, the equation of a given line is represented by $Ax + By + C = 0$ and (x_c, y_c) is the cursor position, then the equation for the normal distance is

$$d = \frac{Ax_c + By_c + C}{\sqrt{A^2 + B^2}}$$

In terms of end points of a line, this equation can be rewritten as

$$d = \frac{(y_1-y_2)x_c + (x_2-x_1)y_c + x_1y_2 - y_1x_2}{\sqrt{(y_1-y_2)^2 + (x_2-x_1)^2}}$$

If there are large number of lines displayed on the screen and, consequently, a large number of such tests will be required and this may become rather time consuming. Therefore, to avoid such computations a better and efficient algorithm for searching lines has been used. The algorithm is given in Chapter 4.

3.4.1.2 Deletion of a line

After having searched the line shown in Fig. 3.6(a), the program counter PC will be pointing to the end point record of the line in the data structure as shown in Fig. 3.6(b). In order to delete this line the delete flag of the record pointed by PC has to be set to true (T). But this may not hold good always. The way in which we are representing lines in the data structure, there are four different possible cases.

Case 1 : In this case the line to be deleted is an isolated one. For this case the action to be taken to delete is as shown in Fig. 3.7(a). In this case the record just below the record pointed to by PC may be a symbol record or already deleted record or starting point of another polygon or may not be any record at all (end of data structure).

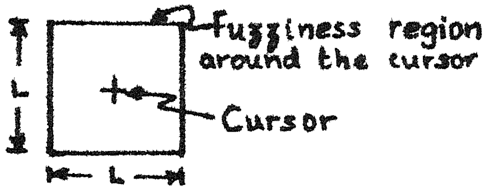


Fig 3·4 Fuzziness region around the cursor for finding a point.

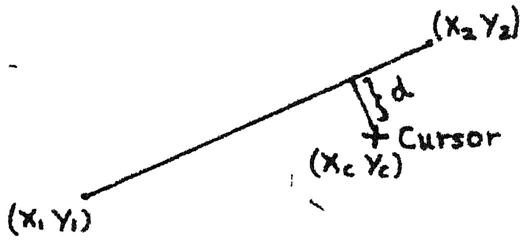


Fig 3·5 Line identification by measuring the normal distance between the line and the cursor position.

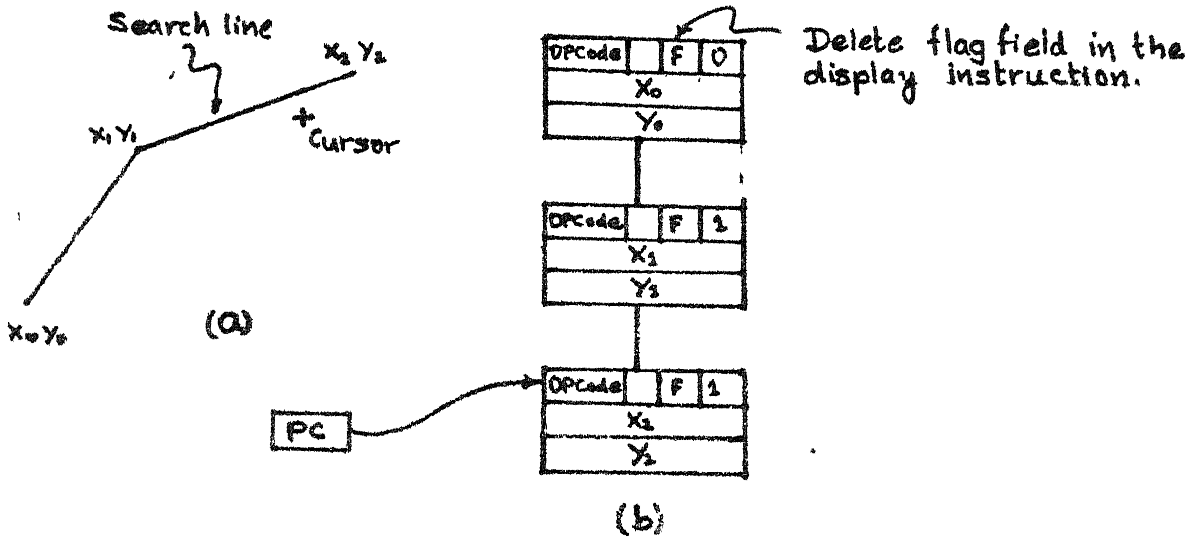


Fig 3·6 (a) Line to be searched in the drawing.
(b) Position of PC After searching in the data structure.

Case 2 : In this case the line to be deleted could be starting line of any polygon as shown in Fig. 3.7(b).

Case 3 : The line to be deleted may be ending line of any polygon as shown in Fig. 3.7(c). In this case the record next to the one pointed by PC may be as in Case 1.

Case 4 : In this case the line to be deleted is in between the two lines of a polygon as shown in Fig. 3.7(d).

3.5 SYMBOL EDITOR

This routine makes use of the low and high limits of the symbol stored with the symbol in the data structure. To search a symbol, the user have to position the cursor anywhere within the low and high limit, which forms a rectangular box. An example of searching an arc is shown in Fig. 3.8. The symbol search routine, for each symbol compares the coordinates of the cursor position with the bounding box. If the cursor lies within the bounding box of the symbol, the searched symbol is indicated to the user. If the symbol is a circle, then the cursor is flashed between centre and circumference else for an arc the cursor is flashed between starting point and centre. Then the user can indicate his choice, whether to delete or to continue searching or to exit.

Deletion is very simple in this case. The delete flag of the records of the symbol have to be made true.

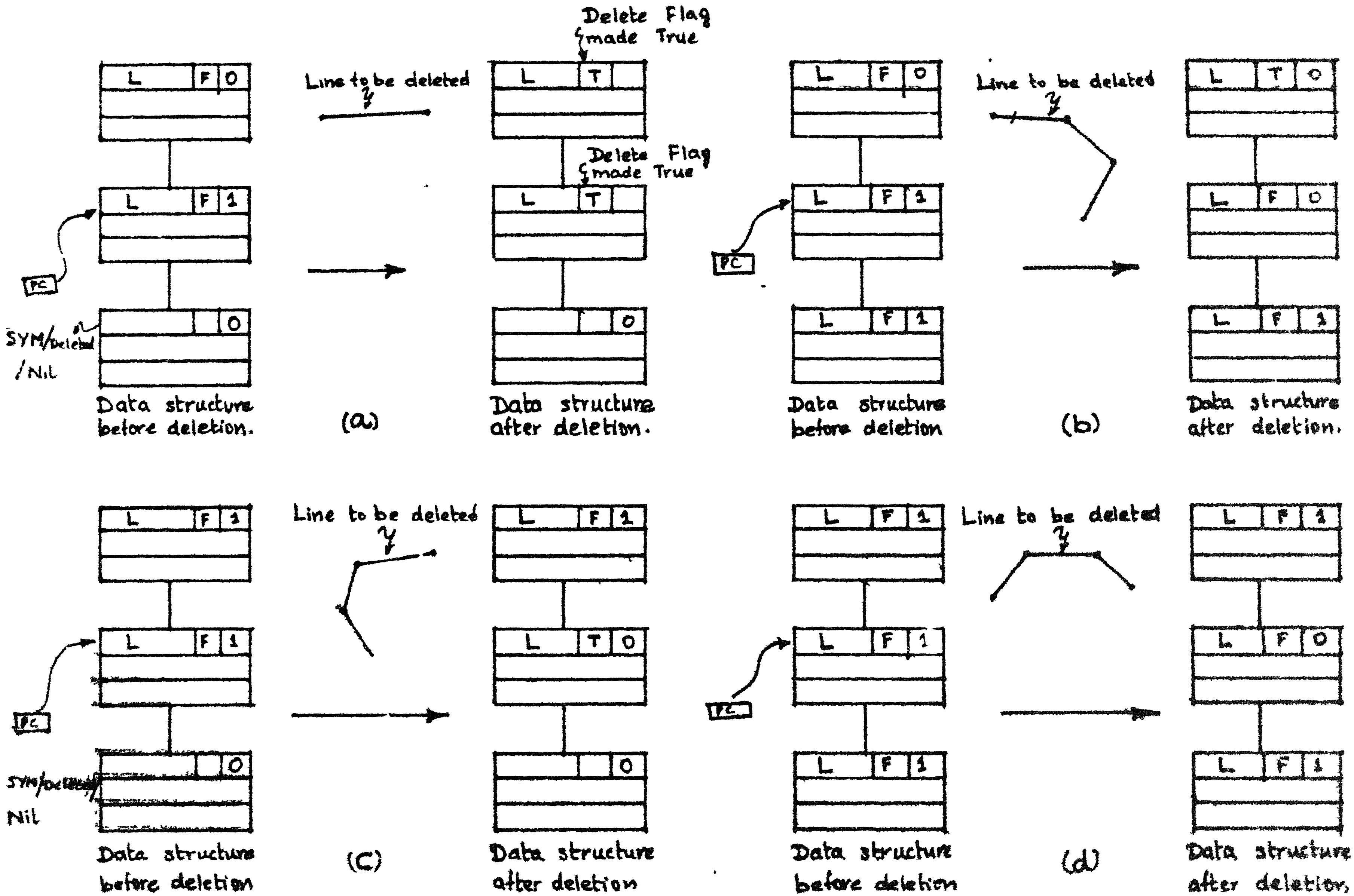


Fig 3·7 Status of data structure after and before deletion of lines.
(a) case 1 (b) case 2 (c) case 3 (d) case 4

3.6 LEVEL EDITOR

In this case all the records in the user indicated level are deleted.

3.7 MACRO EDITOR

This routine is similar to symbol editor wherein the cursor position is compared with the bounding box of the macro. For deleting a macro, the delete flag of all the records belonging to that macro are made true.

3.8 FILER

This routine stores all the drawings created by the user on the disk storage and also maintains the directory of drawing names for each file. The structure of the file storage is as shown in Fig. 3.9. The head of the file contains the directory of all the drawings stored in that file. Each record in the directory has three fields. The first field contains the name of the drawing, the second field is a pointer to the starting record of the display code of that drawing in the file and the third field points to the end record of the display code. Thus the size of each record is 3 words, which is same as that of the display code; only the first record in the directory is different from others. It contains the file identification mark in the first field, the second field is not used and the third field contains the size of the directory. This routine also contains procedures to retrieve, update and delete the drawing from the file.

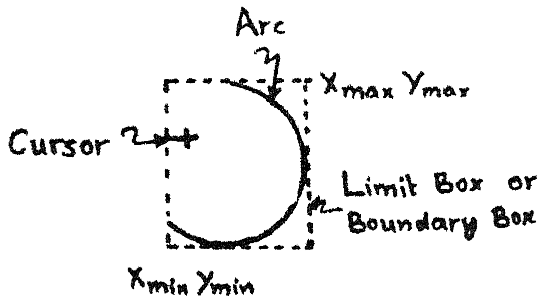


Fig 3·8 Searching for an arc showing the position of cursor, bounding box and the arc.

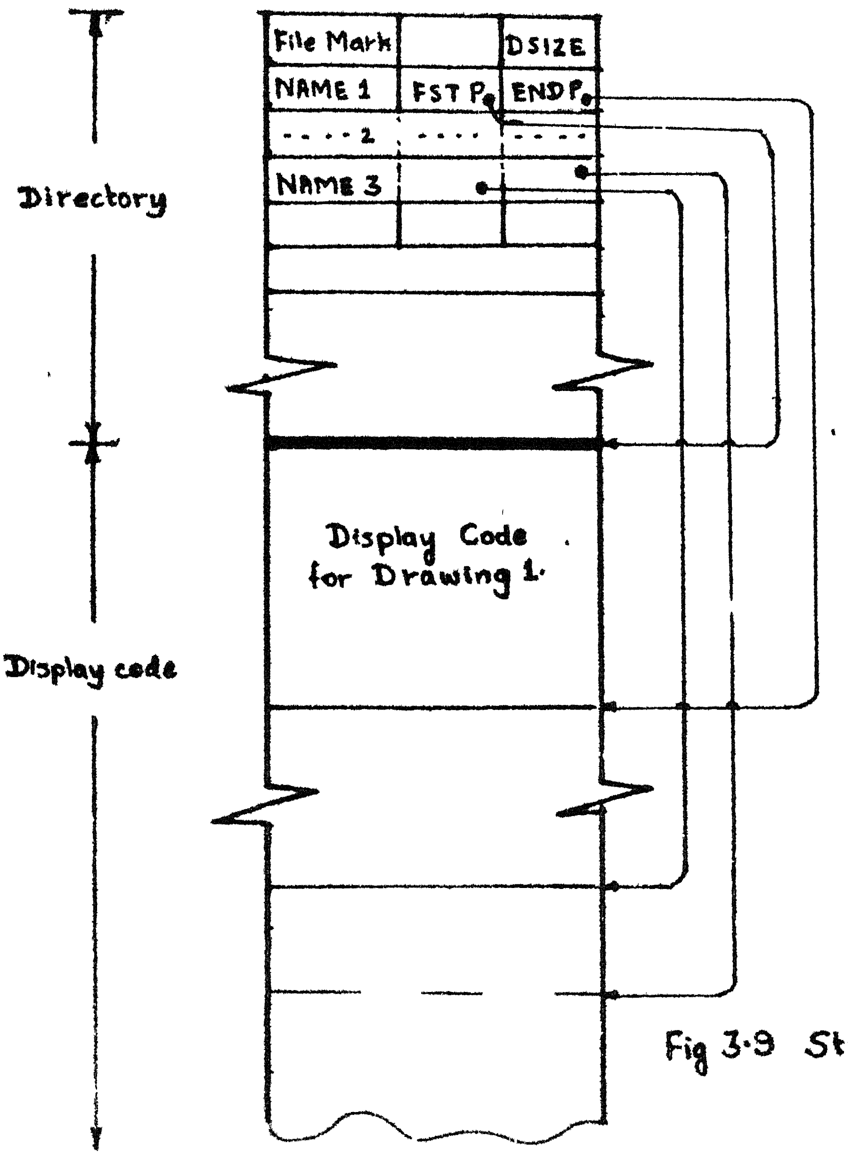


Fig 3·9 Structure of the file storage

3.9 HATCHER

This routine cross hatches the area bounded by a polygon at a given angle. The input to this routine is a list of edges of a polygon to be cross hatched and the angle of hatching. Basically this routine computes the coordinates of a series of lines which start and finish on the boundaries of the polygon at a given hatch angle. In otherwords, it calculates the coordinates of intersections between edges of a polygon and hatch lines.

The idea behind cross hatching a polygon is simple. It has been adopted from the scan conversion procedures of polygons used in raster graphics [2]. Consider a closed polygon as shown in Fig. 3.10. It can be seen that the number of intersections of a line with the closed polygon is always even. In case of hatch line Y = 8(L1), we find two intersections, namely, x = 2 and x = 7. The line between these two points is what we exactly want in hatching. If there are more than two intersections of a hatch line with the polygon then these intersections must be sorted by x value to determine the regions of the hatch line that lie inside the polygon. For example, we find for hatch line y = 4 four intersections at x = 2,3,6 and 7. Sorting in x results in the list (2,3,6,7). Adjacent intersections are then paired together, giving two pairs (2,3), (6,7); each pair represents a region of the hatch line that lie inside the polygon.

3.9.1 Singularities

The method described above works well as long as a vertex of a polygon does not lie exactly on the hatch line. When the vertex of a polygon lies exactly on the hatch line, it is crucial for the proper number of intersections of the polygon boundary and the scan line to be recorded. It is only by counting intersections that the method described above determines which region of the hatch line lies inside the polygon. Fig. 3.11 illustrates two cases of such singularities. It is essential thq that one intersection be recorded for (2,6) and either two intersections or none be recorded for (8,6). Failure to treat these vertices properly will cause improper hatching.

An algorithm for a reliable way of processing such singularities, by computing in an efficient way of intersections with the edges of a polygon and sorting the intersection values computed, is given in Chapter 4.

3.10 WINDOW CLIPPER

Before converting the drawing to be displayed from user coordinates to screen coordinates, all the display information goes through the window clipping routine. This routine CLIPWINDOW uses the standard Cohen-Southerland line clipping algorithm [2]. The lines are clipped to the present user window.

3.11 DISPLAY CODE INTERPRETER

This consists of a set of routines to generate all the premitives mentioned in Chapter 2.

3.11.1 Line Drawing Routines

This consists of three procedures, namely, DOTTED LINE, CHAIN LINE and ARROW LINE. All these lines are drawn according to the I.S.I. recommendation for general engineering drawings as shown in Fig. 3.12 [10].

The procedure DOTTED LINE automatically splits the line into short segments, closely and evenly spaced. Similarly, the procedure CHAIN LINE splits the line into alternate long and short segments in the ratio 6:1. The procedure ARROW LINE appends arrow to the line at the desired ends.

3.11.2 Circle Generating and Arc Generating Routines

This set basically consists of four procedures : DRAWARC, DOTTEDARC, CHAINARC and ARROWARC. The procedure DRAWARC, draws a Circular arc of given radius at any given angle. (360^{o} in case of a circle). This makes use of an efficient circle drawing algorithm described in Chapter 4. All the other arc drawing routines use this algorithm. The procedure DOTTED ARC automatically splits the arc into small evenly spaced arc segments. The procedure CHAINARC splits the arc into long and short arc segments and the procedure

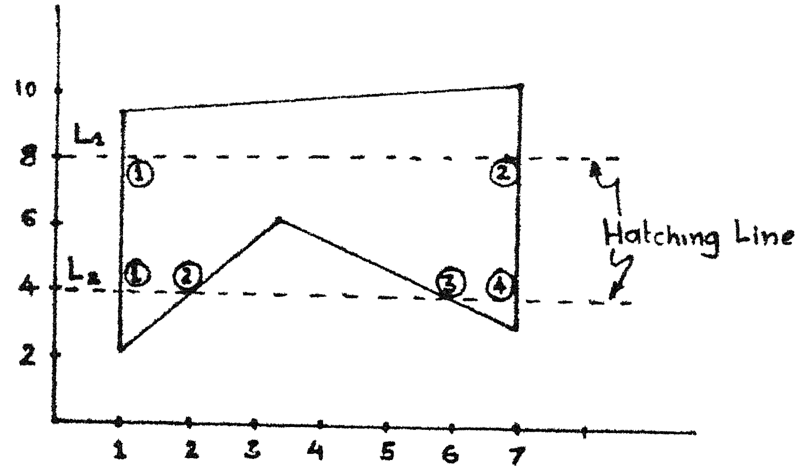


Fig 3.10 Polygon intersecting with hatch lines at even number of points.

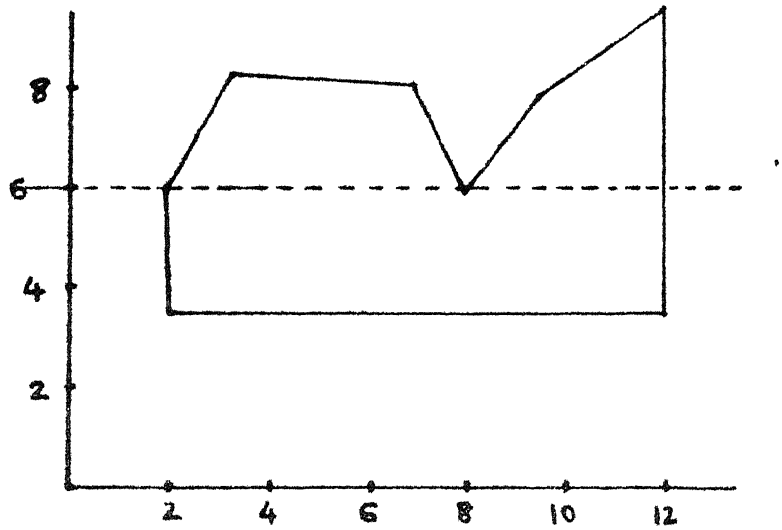


Fig 3.11 Polygon vertices that lie exactly on a hatch line require careful determination of intersections. Vertex at (2,6) generates one intersection while that at (8,6) must generate either two or none.

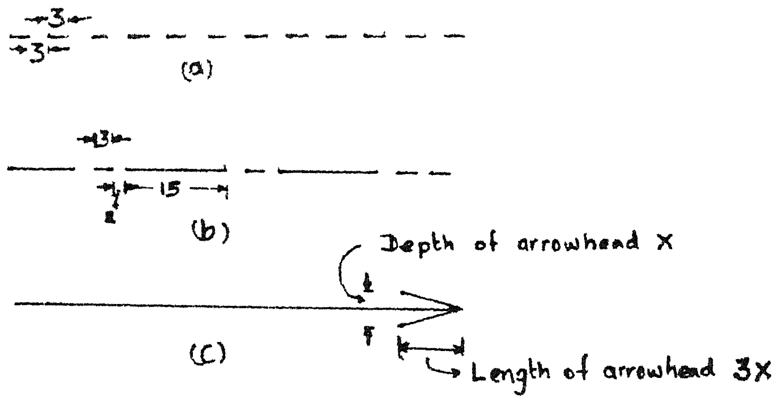


Fig 3·12 Types of lines (a) Dotted line, short dashes closely and evenly spaced. (b) Chain line, alternate long and short lines in the proportion ranging from 6:1 to 4:1. (c) Arrow line, length to width proportion 3:1.

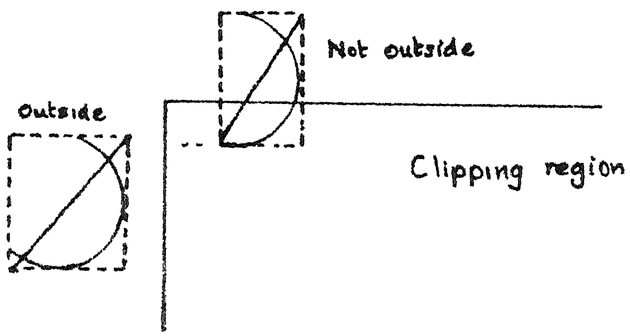


Fig 3·13 Applying the boxing test, by checking a diagonal of each symbol's bounding box for trivial rejection

ARROWARC appends arrows at the desired ends.

3.11.3 Symbol Level Clipping

The symbol drawing routines such as circle and arc routines described above, draws the symbol as a small line segments. Thus each symbol consists of many such line segments. Each line segments before displaying will have to be passed through the window clipping routine. The whole process will be wasteful if the entire symbol lies outside the clipping region. Instead of applying a visiability test to every line segment in the symbol, one should be able to test the whole symbol in a single step. This can be done by incorporating such a test in the display code interpreter. The interpreter, before calling the display procedure representing a symbol, checks whether the entire symbol lies outside the clipping region; if it does, it proceed no further with the procedure call. Here, we use the kind of visiability test called Boxing [2]. This makes use of the bounding box information of the symbol stored in the data structure. This boxing test is very simple. We need only apply the cCohen-Sutherland clipping algorithm to a diagonal of the bounding rectangle, and if it is trivially rejected, we know that the rectangle lies entirely outside the clipping region. This is illustrated in Fig. 3.13. If the symbol lies partially outside the visibility region, then the invisible portion of the symbol is rejected by the line clipping routine.

3.12 TEXT

Since the Tektronix 4006-1 storage-tube display does not have the capability of displaying text strings of different sizes, only strings of fixed size (only in horizontal direction) can be used for annotating the drawings. No attempt has been made to write software character routines. Instead, the hardware character writing capability of the terminal has been made use of.

3.13 EXTERNAL PROCEDURES USED

Since certain I/O facilities for effective interaction are not provided by the language PASCAL, few procedures mentioned below are written in MACRO-10 (DEC-1090 system)[10].

i) INCHRW (Var CH)

This procedure accepts a character from the TTY, as soon as the key is pressed.

ii) SKPINC

This is a function which will be true if there is any character in the TTY buffer.

iii) WAIT (T)

This procedure makes the running program to sleep (inactive) for the time duration specified.

WAIT and SKPINC are used in flashing a cursor by the editor to indicate the searched line or symbol.

iv) OUTCHR (I;: integer)

This procedure outputs on the TTY the ASCII character equivalent to the integer value specified.

Only these procedures and functions have to be re-written to suit a particular mainframe machine.

CHAPTER 4

ALGORITHMS

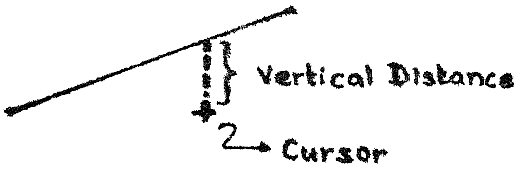
In this chapter some of the important algorithms used in the graphics package are described in detail.

4.1 LINE SEARCHING ALGORITHM

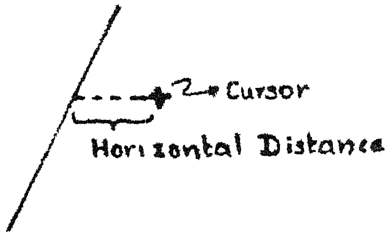
In this algorithm instead of computing the normal distance between the line and the cursor, we compute either horizontal or vertical distance between the line and the cursor, depending upon the absolute value of the slope of the line. If the absolute value of the slope is less than one then we compute vertical distance else horizontal distance as shown in Fig. 4.1.

We identify the line if this horizontal or vertical distance is less than or equal to the maximum search distance specified in the system. This maximum search distance will form a search area around the line as shown in Fig. 4.2. Therefore, the line is identified if the cursor is placed anywhere within this area.

Before calculating the horizontal or vertical distance, lines can be trivially rejected if the cursor does not lie within the trivial rejection region as shown in Fig. 4.3. This can be tested as follows :

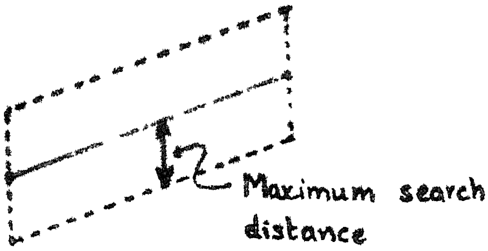


(a)

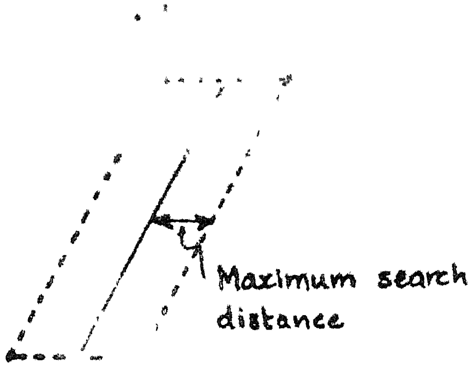


(b)

Fig 4.1 Measurement of distance between the searching line and the cursor.
(a) $|slope| < 1$ (b) $|slope| \geq 1$



(a)



(b)

Fig 4.2 Search area shown by dott line.
(a) $|slope| < 1$. (b) $|slope| \geq 1$.

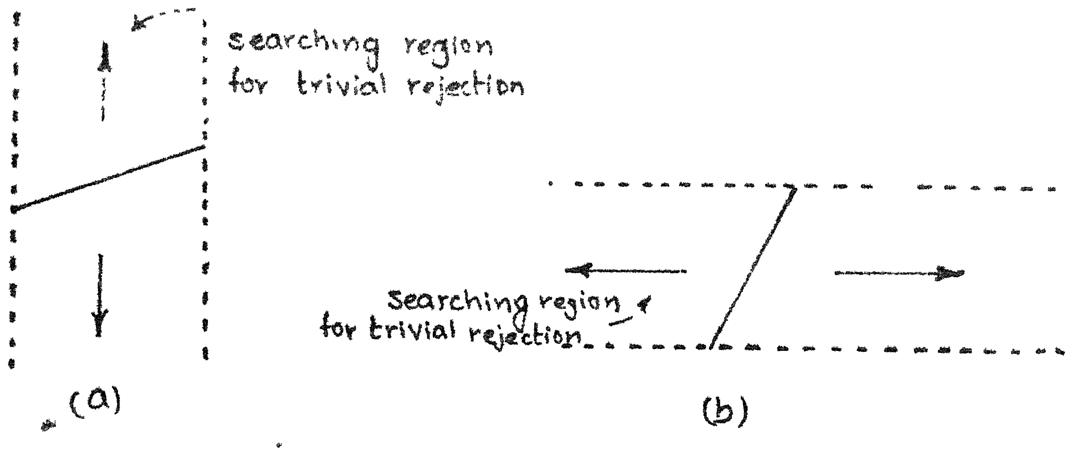


Fig 4.3 Region to be compared with the cursor before trivial rejection.
(a) |slope| < 1 (b) |slope| ≥ 1

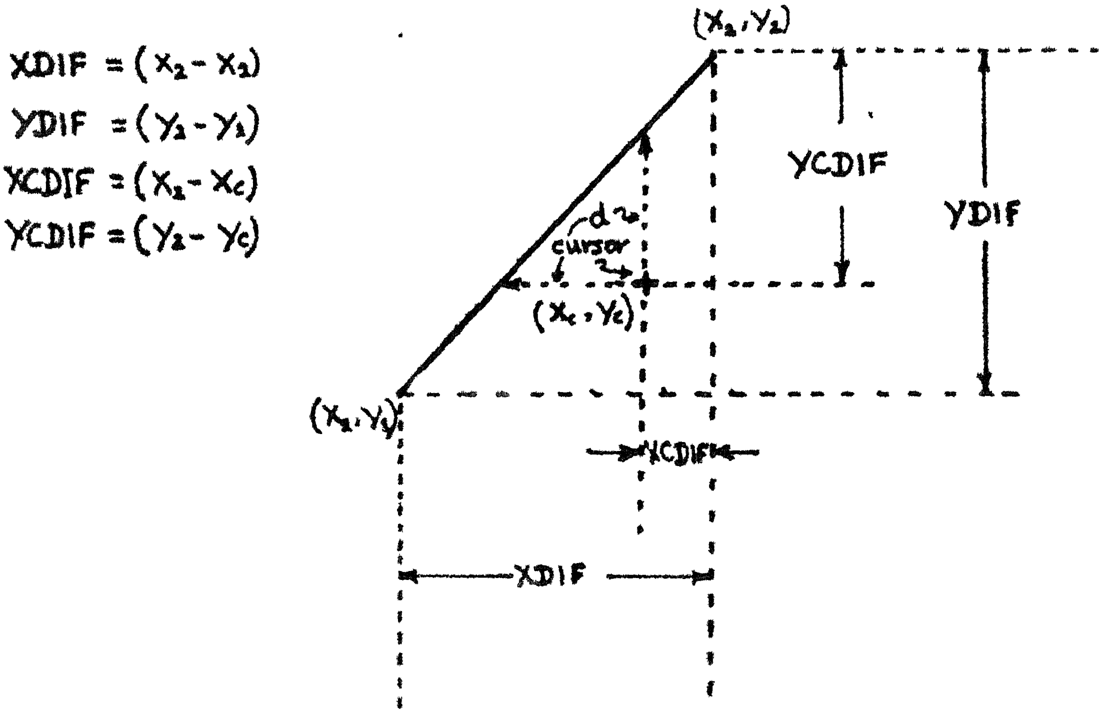


Fig 4.4 Line with different parameters to be computed by the algorithm.

Step 1 : Calculate the absolute slope of the line.

Step 2 : If slope is less than one then calculate

test value = XCDIF/XDIF

else

Calculate test value = YCDIF/YDIF (See Fig. 4.4).

Step 3 : If $0 \leq$ test value ≤ 1 then calculate

horizontal or vertical distance depending upon

the slope

else reject trivially.

If the line is not trivially rejected, calculation of distance is done as follows :

If $|\text{slope}| < 1$ then

Distance = (slope x XCDIF) - YCDIF

else

Distance = (YCDIF/slope) - XCDIF.

If the absolute distance is less than or equal to maximum search distance then the search is successful and that the line is picked. These observations are illustrated by means of a flow chart in Fig. 4.5.

4.2 CIRCLE GENERATING ALGORITHM

A circle can be efficiently represented parametrically by eliminating the necessity for calculating the trignometric functions at each step. This can be accomplished by using the double-angle formulas.

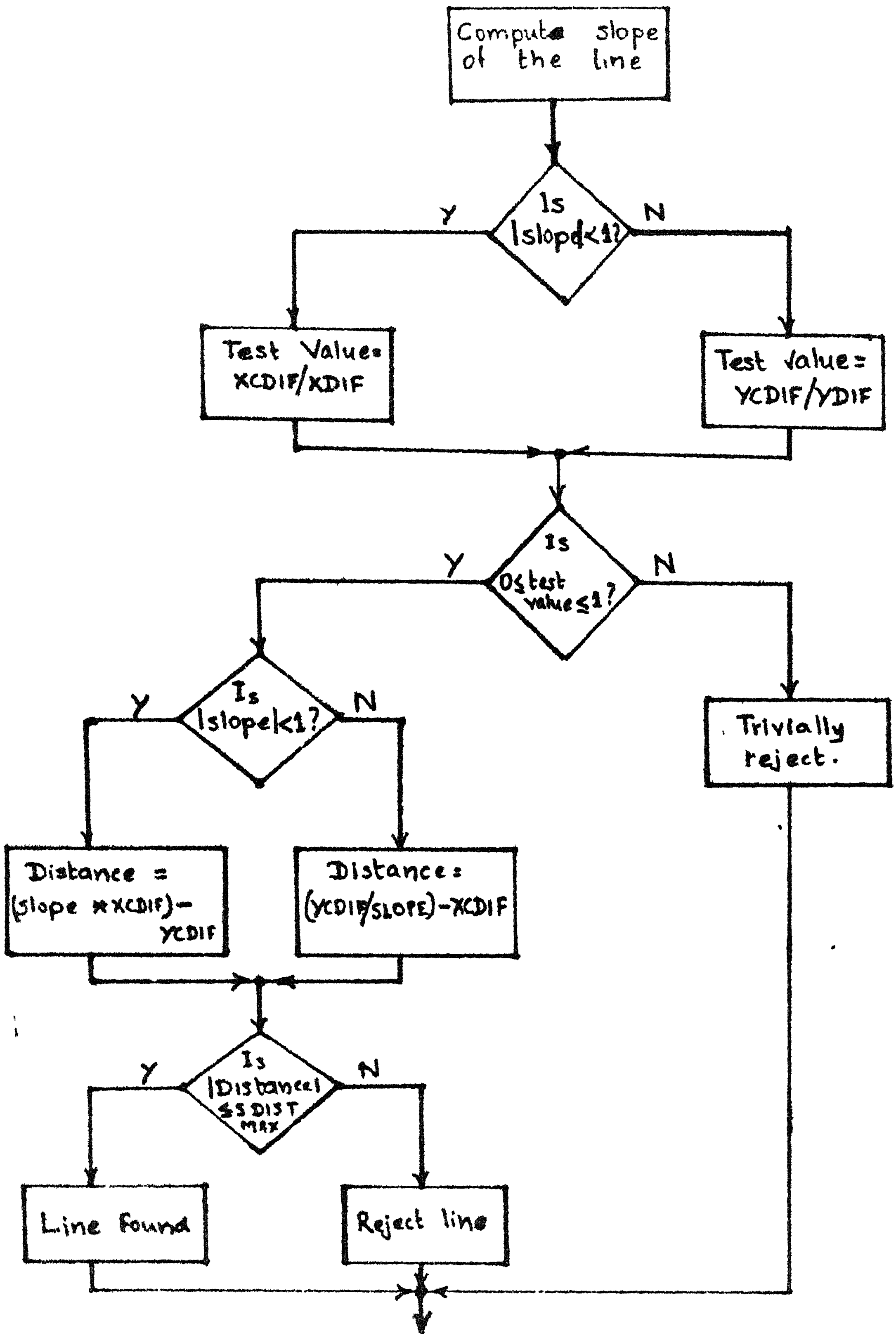


Fig 4.5 Flow chart of the line searching algorithm.

$$\cos(\theta + d\theta) = \cos\theta \cos d\theta - \sin\theta \sin d\theta$$

$$\sin(\theta + d\theta) = \cos\theta \sin d\theta + \cos d\theta \sin \theta$$

Noting that the circle will be completely swept out for a range of θ from 0 to 2π and assuming that a fixed number of points will be used, then $d\theta$ will be a constant. The Cartesian coordinates of any point on an origin-centred circle are then given by

$$x_{n+1} = r \cos(\theta + d\theta) \qquad (4.1)$$

$$y_{n+1} = r \sin(\theta + d\theta) \qquad (4.2)$$

where r is the radius of the circle.

Using double-angle formulas allows eqns. (4.1) and (4.2) to be rewritten as

$$x_{n+1} = x_n \cos d\theta - y_n \sin d\theta \qquad (4.3)$$

$$y_{n+1} = x_n \sin d\theta + y_n \cos d\theta \qquad (4.4)$$

If the circle is located at x_c, y_c relative to the origin of the coordinate system, then we have

$$x_{n+1} = x_c + (x_n - x_c) \cos d\theta - (y_n - y_c) \sin d\theta \qquad (4.5)$$

$$y_{n+1} = y_c + (x_n - x_c) \sin d\theta + (y_n - y_c) \cos d\theta \qquad (4.6)$$

since $d\theta$ is constant; the values $\sin d\theta$ and $\cos d\theta$ need to be calculated only once. This produces an efficient algorithm.

4.2.1 How to Select $d\theta$?

Since we are representing a circle by breaking it up into short lines, $d\theta$ depends on how accurate the circle should be. In otherwords how smooth it should be by appearance. We may just choose a small value of $d\theta$ to give some fixed number of points on the circle (say 100), then there may be more points than needed for circle of 'small' radii and fewer than needed for circles of 'large' radii in order to get an acceptable pPortrayal. Therefore, one would like to choose the number of points (or $d\theta$) on the circle by relating in some prescribed way to the radius. The method we have choosen for calculating $d\theta$ is as follows:

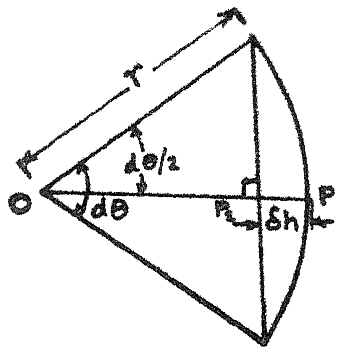
Fig.4.6a shows the linear approximation of the arc of a circle for an incremental value $d\theta$. To establish the relation between $d\theta$ and r, we define smoothness by keeping the difference between OP and OP_1 to a small constant value (δh) which gives an acceptable tolerance between the vector representation of the circle and the true circle. This gives a simple formula for $d\theta$ in terms of r and δh.

$$OP - OP_1 = \delta h$$

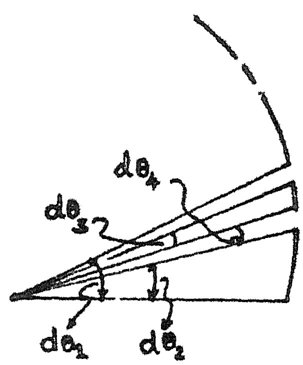
$$r - r\cos(d\theta/2) = \delta h$$

$$d\theta = 2\cos^{-1}\left(1 - \frac{\delta h}{r}\right) \quad \text{for all} \quad \frac{\delta h}{r} \quad 1 .$$

In the present version of the package, the value of δh has been set to 0.01.



(a)



(b)

Fig 4·6 (a) Error in approximating the arc of a circle by a vector.
(b) Division of chain circle into different angled arcs.

4.2.2 Chain Circle

In order to draw a chain circle using circle generating algorithm, we have to compute four incremental values of $d\theta$'s as shown in Fig.4.6b, so that the circle could be broken into alternative long and short arcs. If the $d\theta_1$ and $d\theta_3$ values calculated for drawing these long and short arcs are greater than the value of $d\theta$ calculated for drawing a circle, $d\theta_1$ and $d\theta_3$ may have to be split into integral multiples of new $d\theta$ so that the acceptable arc error is maintained.

4.3 HATCHING ALGORITHM

The algorithm is illustrated by means of a flow chart as given in Fig. 4.7. Each step in the flow chart is described below.

The given list of edges of a polygon to be cross-hatched are first transformed through a clockwise rotation by an angle equal to the hatch angle. This makes the hatching lines horizontal with respect to the x-axis and makes the computation of intersections an easy task. During the transformation process, low and high limits of the transformed edges are found. From this, y_{max} and y_{min} values are used to conclude the starting Y value of the hatch line and the maximum number of hatch lines required in the hatching process. This is computed by having a system-specified distance between the hatch lines. Now, to compute the intersections efficiency

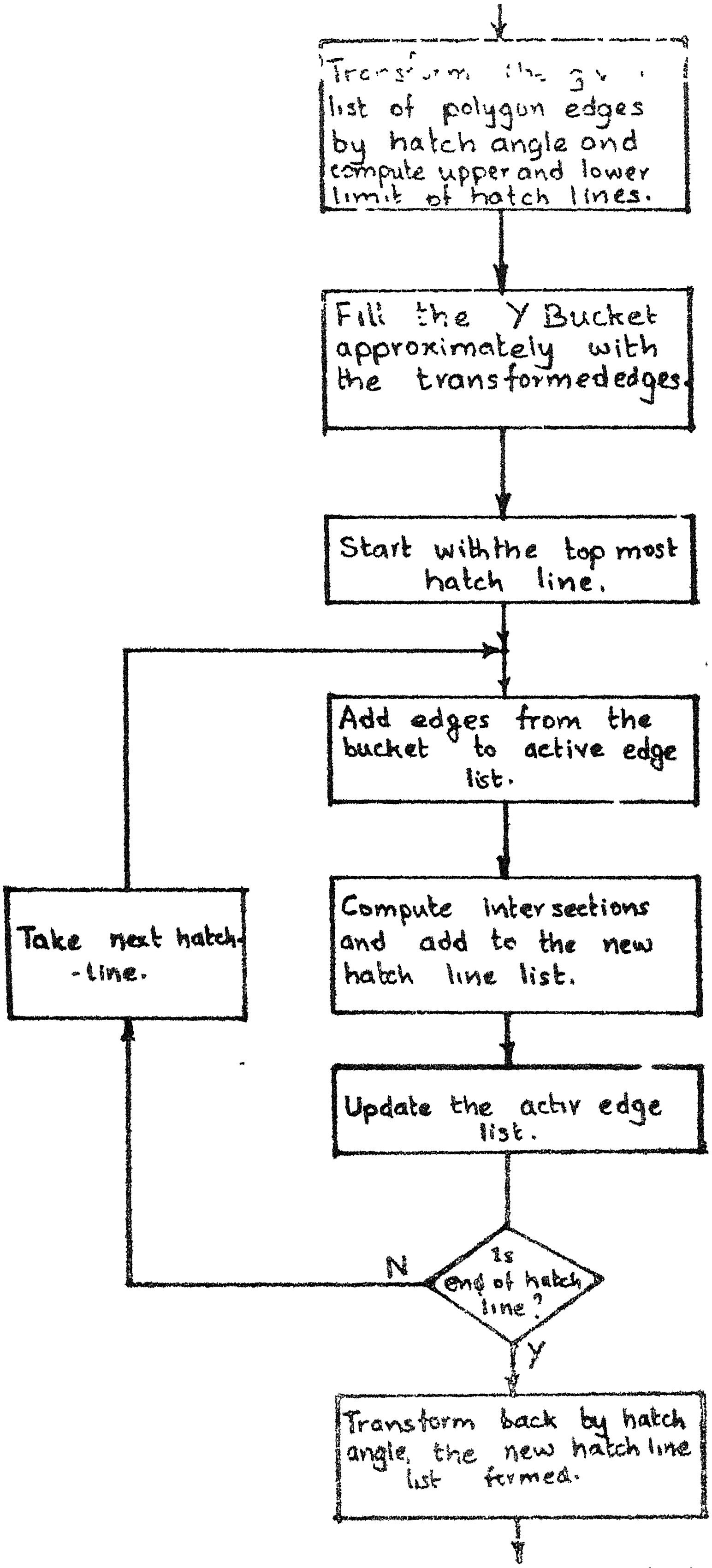


Fig 4.7 Flowchart for the hatching algorithm.

for each hatch line with the transformed edges (from now onwards we call just edges) of the polygon, we need an approach that generates intersections only as they are needed, one hatch line at a time. For each hatch line, we must first compute the intersections that lie on that hatch line; then these intersections are sorted by x coordinate to determine the intersections that are to be paired together.

The principal data structure used to do this is a list of polygon edges, sorted by maximum Y value. This is called the Y-bucket list, because it is generated by a bucket sort in which each edge is entered in the 'bucket' corresponding to its maximum Y value (Fig. 4.8). Each edge record contains the information needed to generate intersections: x, the location of the intersection on the topmost hatch line; dx, the amount by which the x value changes from hatch line to hatch line; and dy, the number of intersections that the edge will generate. This process of filling the bucket is done by the procedure 'FILLBUCKETLIST' in the program. This procedure checks for the singularities that may arise as mentioned in Chapter 3.

A reliable way of avoiding singularity is given below :

This algorithm is based on the topology of the polygon. It makes use of the direction of successive edges of the polygon. If the polygon boundary progresses monotonically

upward or downward, only a single intersection with each hatch line should be recorded. When the direction changes from downward to upward or from upward to downward, an intersection must be repeated. This is illustrated in Fig. 4.9. The vertex (4,8) in the figure generates one intersection because edges progress monotonically through the vertex, whereas the vertex (8,8) generate two intersections because the vertical direction of the boundary changes at the vertex. These observations are done while entering the bucket list by the following algorithm, which also checks for horizontal edges properly. This maintains a singularity list in which each record contains the (x,y) value of the vertex which lie on the hatch line; topology of the edge through that vertex, i.e., up or down and pointer to the next record in the list.

Algorithm

1. Retrieve the next edge of the polygon. If there are no more edges, exit.

2. If this edge has no intersections with the hatch lines it is horizontal or very nearly so. Go to Step 1.

3. Calculate the first Y intersection, the number of intersections dy, the location of the intersection on the topmost hatch line x and the increment for each hatch line dx. From an edge record.

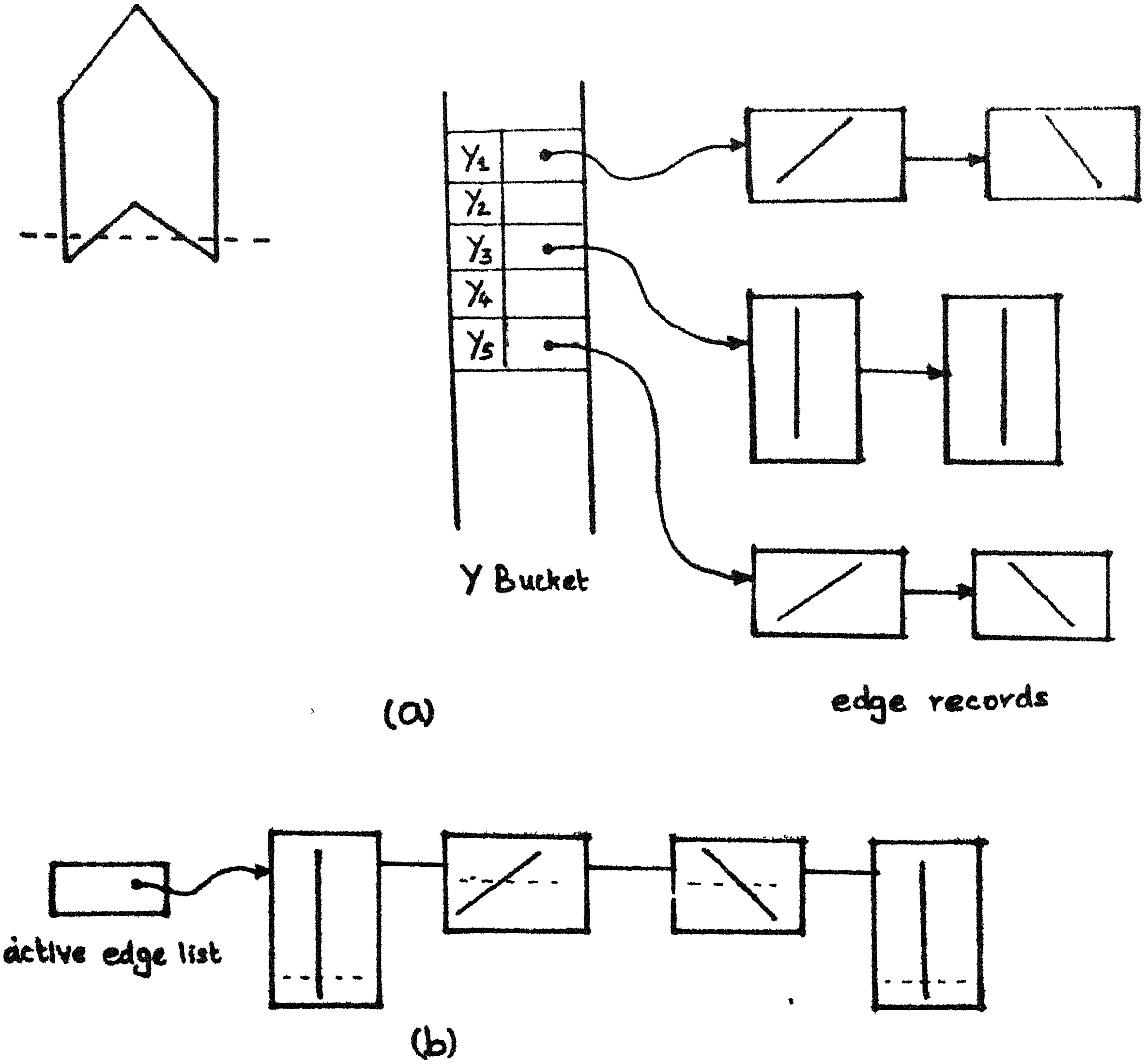


Fig 4·8 Schematic representation of the data structure for the hatching algorithm. (a) a polygon and associated Y-bucket list of edge information; (b) the active edge list as it appears while processing a hatch line (shown by dotted line)

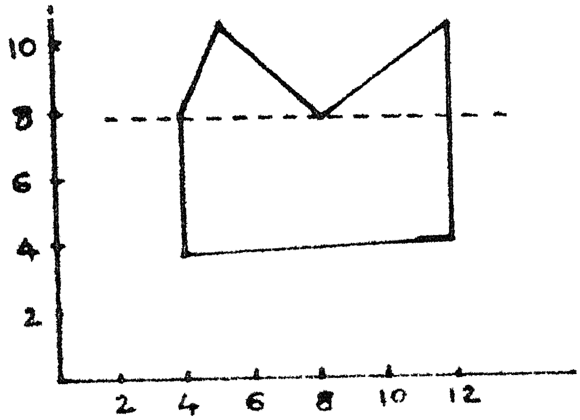


Fig 4·9 Polygon vertices that exactly lie on a hatch line.

4. Check for singularity by finding any intersections with the edge vertex. If there is any then find the topology of the edge. Check the singularity list for any vertex with the same value. If there is any then check the topology. If the topology is same then if it is down then decrement dy by one and first y value by one hatch line distance. If it is up then decrement dy by one. If no vertex of that value is found add the present vertex to the singularity list with the topology.

5. If dy is not zero then add this edge in to the bucket list corresponding to first y value of this edge. Go to Step 1.

After having filled the bucket, intersections are generated one hatch line at a time, starting from the top hatch line. As it does so, it maintains an active edge list, sorted by x, listing all edges that cross the hatch line being prepared (Fig. 4.8b). Successive pairs of these edges demarcate portions of the hatch line which lie inside the polygon and used to fill the new list of vectors which represent the hatch line to be drawn. After filling the new list for this hatch line, each edge record is updated for the next hatch line (i.e., x = x+dx; dy = dy-1). These updates may occassionally require edges in the active edge list to be interchanged so that the list remains sorted

by x. Some edges will terminate (dy = 0) and be removed from the list. Any edge record cited in the y bucket for the new hatch line need to be added.

It should be noted that the effort required to keep the list sorted is very small because of the relatively low frequency with which edges cross, necessitating changes in the order. Therefore, bubble sort is done on the list. While inserting new edge records to the active edge-list, insertion sort is done.

After forming a new list of hatch lines, all these lines are transformed back by the same hatch angle and then displayed.

CHAPTER 5

RESULTS AND CONCLUSIONS

5.1 ILLUSTRATIVE EXAMPLES

In order to show how the various facilities provided in the package can be utilized, a set of three examples have been selected. The results are as follows.

Example 1 : The three orthographic views - the top view, the front view and the side view - of a slide block have been created and shown in Figure 5.1. Table 5.1 gives a listing of commands used for creating the front view. Similar commands have been written for creating the other views. Once a view is created then the axis lines are drawn. Finally, the dimension lines are drawn along with arrowed ends and then all the dimensions are written down as character strings. Note that all the instructions are given in the user's world coordinates. This facilitates easy conversion from a sketch to a drawing. In the listing given in Table 5.1, one can see the usage of FIND commands for easy location and pointing of centres of the semi-circles.

Example 2 : This example illustrates the usage of hatching facilities. Figure 5.2 shows two orthographic views of a Solid Bearing. Since the front view is symmetric about the central vertical axis line, it is sufficient to have a half-sectional view. The portion sectioned has been assigned a

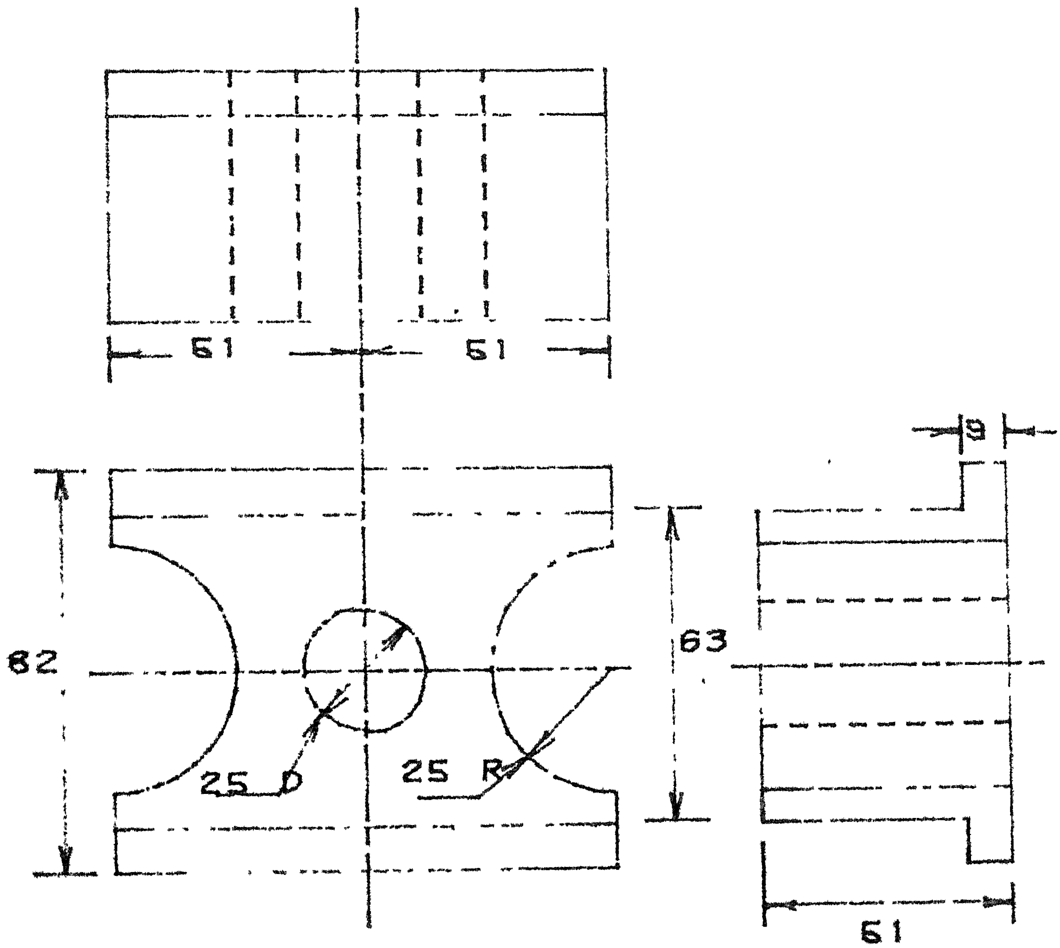


FIG.(5.1) SLIDE BLOCK SCALE 1:10

separate level and the entire region can then be hatched with hatch lines drawn at an angle of 45 degrees. As in the previous example, a set of commands have been prepared for this example from a rough sketch of the two views of the bearing.

Example No.3 : This example illustrates the usage of macro facilities. Figure 5.3 shows some views of a Lap Joint. In this case, a series of rivets are to be shown so as to indicate the arrangement of rivets. This can be achieved by creating a macro showing a view of only one rivet and then the macro can be posted thrice so as to show an assembly of three rivets. Similarly, if a joint has two rows of rivets then the front view can also be constructed by having a macro showing the front view of one rivet and posting it twice side by side. If successive postings have any undesirable lines then these can be deleted from the drawing without altering the definition of the macro.

5.2 TECHNICAL SUMMARY

In the present work, an attempt has been made to explore some of the issues involved in developing a simple, general purpose, engineering drafting package. In the present work, a simple and compact data structure has been used to represent and store graphic entities.

A variety of features like different types of editing features, the hatching facility, the facility to structure

the data into levels and the filing facility has enhanced the capabilities of the package. A careful separation of the device dependent features from the rest of the software extends the use of the package for different output devices (besides Tektronix 4006) with little modification in the package. To achieve machine independence, the entire package has been written in the higher level language PASCAL. Truely speaking, due to variations in the features of several operating systems together with minor differences between language dialects, true machine independence is an almost unattainable goal. Some minor changes in the package are always required when a program is moved to another site.

5.3 LIMITATIONS AND SUGGESTIONS

The present version of the package uses the keyboard as the only input device. The keyboard becomes a very slow input device for feeding in the data of a drawing compared to a tablet or a digitizer. If such input devices are available then the package can be modified accordingly. This will be especially useful for generating large, complex drawings.

The present version of the package is a general purpose drafting package. However, drafting needs vary considerably from one area of applications to another. For instance, in case of electrical drafting the need for a powerful macro facility may be more urgent compared to the hatching

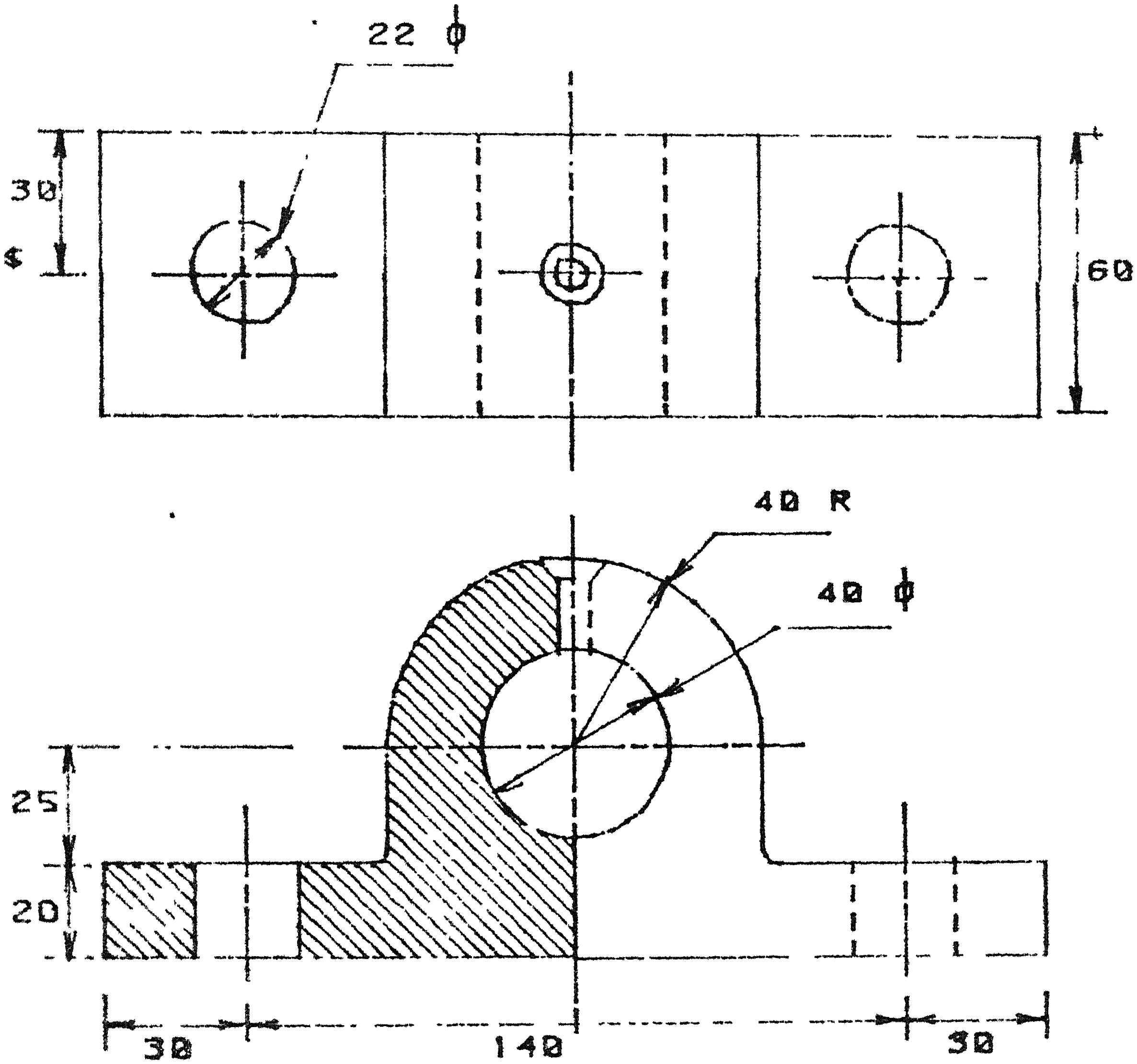


FIG. (5 2) SOLID BEARING SCALE 1:10

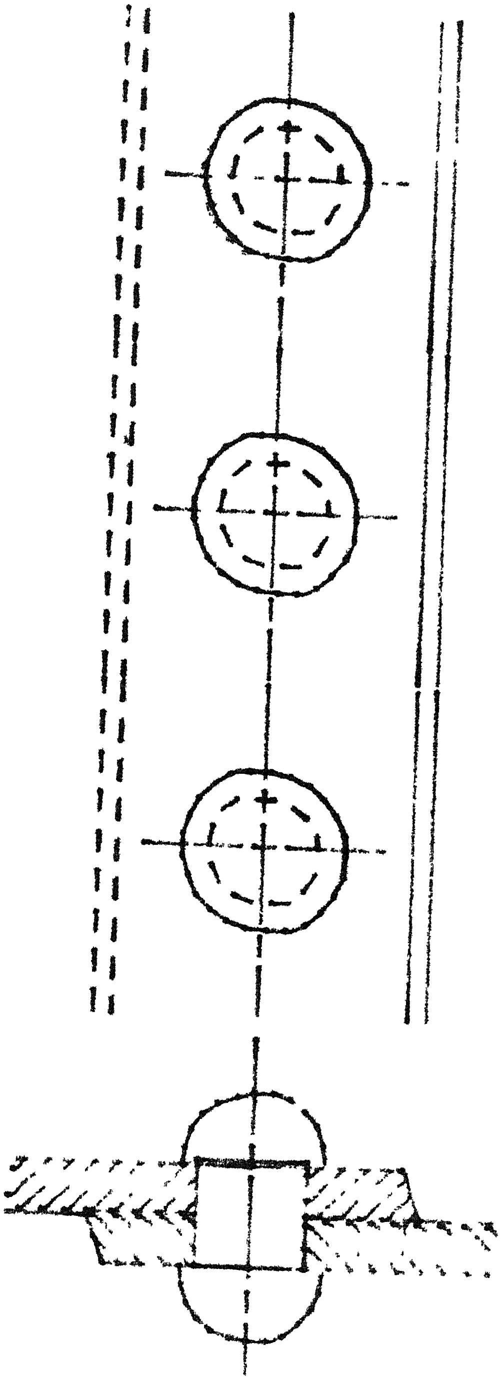


FIG (6.3) SINGLE-RIVETED LAP JOINT

facility. Several codes have been developed for drafting applications in piping layouts, circuit layouts, chemical process layouts and so on. If one needs to use a drafting package for a specific area of applications then one can use the present version and update it so as to fulfil specific needs successfully.

It is desirable to generate software characters for strings instead of hardware characters made available in the present version. This will enable usage of several styles of character fonts as well as effective control on the size of characters. It is also desirable if the process of dimensioning can be made automatic. This can be achieved by setting up specific rules about a layout scheme of dimensioning. In the present package the data structure is maintained only in the main memory. This restricts the amount of drawing information that can be input at any one time to the maximum memory allocated for data structure. However, with the present data structure one can write a sophisticated memory management scheme to maintain a park of the data structure in the main memory and the rest on the disk.

It should be noted that in the present work all the geometrical information is considered to be two-dimensional in nature. Recently, considerable work is being undertaken by researchers in the area of three-dimensional geometric modelling. In geometric modelling emphasis is being placed on

designing or conceptualizing an object rather than drafting it. However, after a geometric model is conceived, it is necessary to draw it. In this context, it will be interesting to explore as to how a drafting package can be used or developed along with a geometric modelling package.

Table 5.1

Part of Command Listing of
Example No.1

```
PC X30Y30

LS L102

LS L16A90

PCR Y:25

ARS A-180

LS L16 A90

LS L102A180

LS L16A-90

PCR V-25

ARS A-180

LS L10 n-90

PCR Y9.5

LS L102

Find left top corner

PCR Y-9.5

LS L102

Find center of right arc

PCR X-51

CS R12.5
```

APPENDIX A

COMMAND SYNTAX AND DESCRIPTION

A.1 SYNTAX

The syntax for the monitor and drafting mode commands are given in BNF notation. In this notation, the symbol < >^{1} means zero or one occurrence that is, an optional occurrence of the object in brackets. Default values are assumed on not specifying such parameters. The notation < >$_1^n$ means one or n occurrences of the object in brackets.

<Letter> :: = 'A' | 'B' | .... | 'Z'

<Digit> :: = '0' | '1' | .... | '9'

<Delimeters>:: = ',' | ':' | Space

<Identifier>:: = <Letter>$_1^5$

<Number> :: = <Real> | <Integer>

<Integer>:: = <Sign>^{1} <Digit>$_1^8$

<Real> :: = <Sign>^{1} <Integer>^{1} '.' <Digit>$_1^5$

<Sign> :: = '+' | '-'

<CTC>:: = CARRIAGE RETURN (command terminating character)

A.1.1 Drafting Commands

i) Lines

<Line Solid> :: = 'LS' <Parameter>^{1} <CTC>

<Line Dotted>:: = 'LD' <Parameter>^{1} <CTC>

<Line Chain>:: = 'LCH' <Parameter>^{1} <CTC>

<Line End Arrow>:: = 'LA' <Parameter>^{1} <CTC>

<Start Arrow Line>:: = 'AL' <Parameter>^{1} <CTC>

<Arrow Line Arrow>:: = 'ALA' <Parameter>^{1} <CTC>

<Parameter>:: = 'L' <Number> <Angle>^{1}

<Angle>:: = 'A' <Number>

ii) Circles

<Circle Solid>:: = 'CS' <Radius>^{1} <CTC>

<Circle Dotted>::= 'CD' <Radius>^{1} <CTC>

<Circle Chain>:: = 'CCH' <Radius>^{1} <CTC>

<Radius>:: = 'R' <Number>

iii) Circular Arcs

<Arc Solid>:: = 'ARS' <Angle> <CTC>

<Arc Dotted>:: 'ARD' <Angle> <CTC>

<Arc Chain>:: = 'ARCH' <Angle> <CTC>

<Arc End Arrow>:: = 'ARA' <Angle> <CTC>

<Start Arrow Arc>:: = 'AAR' <Angle> <CTC>

<Arrow Arc Arrow>:: = 'AARA' <Angle> <CTC>

<Angle>:: = 'A' <Number>

iv) Cursor Commands

<Position Cursor Absolute>:: = 'PC' <Coordinates> <CTC>

<Position Cursor Relative>:: = 'PCR' <Coordinate> <CTC>

<Move Cursor by L and A>:: = 'CMOV' <Parameter> <CTC>

<Set Cursor Step>:: = 'STEPC' <Number> <CTC>

<Coordinate>:: = <X-CORD>[1] <Y-CORD>[1]

<X-CORD>:: = 'X' <Number>

<Y-CORD>:: = 'Y' <Number>

v) Editing

<Search Line>:: = 'SLINE' <CTC>

<Search Symbol>:: = 'SSYM' <CTC>

<Search Macro>:: = 'SMAC' <CTC>

<Search String>:: = 'SSTG' <CTC>

<Delete Level>:: = 'DLEV' <Integer> <CTC>

vi) Miscellaneous

<Show Level>:: = 'SLEV' <Integer> <CTC>

<Change Level>:: = 'CLEV' <Integer> <CTC>

<Set Window>:: = 'SWIND' <CTC>

<Default Window>:: = 'DWIND' <CTC>

<Write String>:: = 'STG''Ø' <String> 'Ø' <CTC>

<String>:: = Any ASCII Character String

<End And Quit>:: = 'EQ' <CTC>

<End And Save>:: = 'EB' <CTC>

<Set Hatch Angle>:: = 'HANGL' <Number> <CTC>

<Call MACRO>:: = 'CALL' <Identifier> <Instance>^{1}

<Instance>::=<Scaling>^{1} <Rotation>^{1}

<Scaling>:: = 'S' <Number>

<Rotation>:: = 'R' <Number>

<Hatching>:: = 'HATCH' <Hatch Option> <CTC>

<Hatch Option>:: = 'POLY' | <Level Number>

<Level Number>:: = <Integer>

<Set Macro Origin>:: = 'ORIGN' <CTC>

A.1.2 Monitor Commands

<Drawing Directory>:: = 'DIR' <CTC>

<Macro Directory>:: = 'DIRM' <CTC>

<TYPE Drawing>:: = 'TYPE' <Identifier> <CTC>

<Type Macro>:: = 'TYPEM' <Identifier> <CTC>

<Delete Drawing>:: = 'DEL' <Identifier> <CTC>

<Delete Macro>:: = 'DELM' <Identifier> <CTC>

<Create Drawing>:: = 'DRAFT' <Identifier> <CTC>

<Create Macro>:: = 'MACRO' <Identifier> <CTC>

<Set Input Scale>:: = 'SCALE' <Number> <CTC>

<Kill Job>:: = 'KJOB' <CTC>

A.2 COMMAND DESCRIPTION

A.2.1 Drafting Commands

A.2.1.1 Single character commands

1) Cursor movement

Four keys marked '∧', ']', '<', '>' have been assigned

to move the cursor up, down, left and right respectively in steps of 1/20th of the window size (default value). For fine movement of the cursor the same keys are used alongwith the SHIFT key.

ii) Find Command

The key 'F' finds a point (end points of a line, centres of arcs and circles) in the drawing and moves the cursor onto that point. The cursor should be placed close enough to the point to be searched. If tne point to be found is within a 1 cm square, centered around cursor, then the point is identified and the cursor is moved to that point. If no point is found, a bell is rung to indicate the failure.

iii) Print Cursor Coordinate ('?')

On pressing '?' key, x as well as y coordinates of the current cursor position in user coordinates are printed in the command area.

iv) Fix Cursor Position ('X')

This will fix the cursor position by marking with a '.' on the screen at the current cursor position. The use of this command is explained in line drawing and hatching commands.

v) Redisplay ('R')

As there is no way of showing the modified drawing after

deleting lines or symbols from the drawing without clearing the screen (an inherent problem with DVST terminals), on pressing 'R' key the screen is cleared and the entire updated drawing is redisplayed.

A.2.1.2 Multicharacter Commands

i) Lines

The command verbs of all types of lines are formed from the root 'L'. The type of line is indicated by adding characters to the root. For example, 'S' is added to indicate solid line. 'D' for dotted line, 'CH' for chain line, 'A' at the beginning of 'L' to add arrow at the beginning, and so on. Thus the command can be remembered easily by the user. Lines can be drawn in two ways :

1) by specifying the parameters, that is, exact length (in user coordinates) and angle. This draws the specified line from the current cursor position. If the angle is not specified then default value of zero is assumed.

2) after the command verb if the parameters are absent then the specified type of line is drawn between the fixed cursor position (see fix command) to the current cursor position.

In all the line drawing commands, the fixed cursor position is automatically updated to the end point of the line currently drawn.

ii) Circles

All circle drawing commands start with the root 'C' and the type of circles are indicated by appending the root with the characters similar to lines.

If the radius is not specified then the radius is calculated as the distance between the fixed and current cursor position. The circle of specified type is drawn with the current cursor position as the centre.

iii) Circular Arcs

All arc drawing commands have the root 'AR'. Different type of arcs can be specified in an exactly similar way to the lines. An arc of specified type and angle is drawn from the fixed cursor position with the current cursor position as the centre. Therefore, before giving the command it is necessary that the starting point of the arc has to be specified by using fix command and then moving the cursor to the centre of the arc, to be specified.

iv) Cursor Commands

Cursor can be positioned exactly by specifying the x and y coordinates. The command 'PC' will position to the absolute coordinates specified whereas the command 'PCR' will position relative to the current cursor position. Specified coordinates can be either x or y or both. The command 'CMOV' will move the cursor by specified length and angle from the current cursor position. The command 'STEPC' will set the step size of cursor movement to the specified number.

v) Editing Commands

The command 'SLINE' searches for the line near to the current cursor position (within 1 cm horizontal or vertical distance). If any line is found, then the cursor is flashed between the endpoints. Now, the user can delete that line by pressing 'D' or continue searching by pressing 'C' or stop searching by pressing 'S'. Bell is rung if no line is found. The command 'SSYM' searches for the symbols (arcs and circles). The cursor has to be placed within the low and high limits of the symbol to be searched. If the search is successful then the cursor is flashed between the centre and circumference for a circle and between centre and starting point for an arc. The user can indicate his choice for further interaction by pressing 'D' or 'C' or 'S'.

Similarly, 'SMAC' searches for the macro and 'SSTG' searches for the string. Strings are searched by placing the cursor near the starting point of the string (within 1 cm). The command 'DLEV' deletes all the drawing information in the specified level.

vi) Miscellaneous

'SLEV' (show level) displays all the information in the specified level. 'CLEV' change the current level to the specified level. When the system is started, a default value of one is set as current level.

'SWIND' sets the user window with the lower limit as the fixed cursor position and the higher limit as the greater of the difference between x and y coordinates of the current cursor position and the fixed cursor position.

'DWIND' will reset the user window to default value of x_{min}, y_{min} to zero and x_{max}, y_{max} to 24.4 x scale (length of the plotter x scale).

'STG' draws the string specified between '\$' signs, with the starting point of the string as the current cursor position. 'EQ' will terminate the drafting mode without saving the drawing on the disk. If the current drawing is already existing on the disk, no modification is done. 'EB' will terminate the drafting mode and save the drawing on the disk. If the old copy of the drawing is existing on the disk it is replaced by the new drawing.

'HANGL' sets the hatch angle to the specified value. The default value at the time of start is 45^{o}. 'HATCH' will hatch the specified area at an angle already set. The area to be hatched can be specified in two ways :

1) after the command verb 'HATCH' if a level number is specified then the drawing area under that level is hatched. This is very useful if the area to be hatched consists of arcs and circles.

2) after the command verb 'HATCH' if the word 'POLY' is followed then the system requests the user to mark the vertex of the polygon to be hatched. The vertex can be specified by first placing the cursor at the required position by cursor moving keys and then pressing the key 'X'. To locate exactly a point (end-points of a line) already existing find command 'F' can be used. The end of polygon is indicated by pressing the key 'E'.

After hatching is done, the system requests the user to indicate whether the hatching is done properly or not. If the user is satisfied, on pressing 'Y' key, the hatch lines computed are stored. Pressing 'N' key will not store the hatch lines.

'ORIGN' will set the present cursor position as the origin of the macro. This is set before storing the created macro into the macro library. 'CALL' will place the called macro from the macro library into the current drawing area after applying the specified scaling and rotation (with respect to the origin of the called macro). The current cursor position is taken as the origin for the called macro. In the absence of scaling and rotation default values of one and zero are assumed respectively.

A.2.2 Monitor Commands

i) Directory Command

The command 'DIR' lists the directory of the drawing names stored in the drawing file. 'DIRM' lists the directory of the macro names stored in the macro library.

ii) Type Command

The command 'TYPE' displays the contents of the drawing name specified in the command on the screen. 'TYPEM' displays the specified macro from the macro library on the screen.

iii) Delete Command

'DEL' deletes the name of the drawing specified from the drawing file. Similarly, 'DELM' deletes the specified macro from the macro library.

iv) Set Scale

The command 'SCALE' sets the input scale to the specified value. This scale value signifies the scale to which the drawing inputted to be plotted on the plotter. For example, if the scale value of 10 is specified, the drawing is plotted with a scale 1:10, that is, one centimeter on the plotter corresponds to 10 units on the drawing.

v) Drafting Command

The command 'DRAFT' will allow the user to draft the drawing with the specified name. This command switches the mode from monitor to drafting. If the specified name is already existing then it draws the existing drawing on the screen and allows to edit.

The command 'MACRO' will allow the user to create a macro with the specified name. If the specified macro is already existing then the macro is displayed on the screen for editing.

A.3 ERROR MESSAGES

1) / CMD ERROR

A command string that cannot be recognised is found.

2) /. Illegal char in file name

Characters which are not alphanumeric or '.' were found in the file specified.

3) % LEN NEG

Length is found negative.

4) % RAD NEG

Radius is found negative.

5) / NUM EXP

Real or integer number expected.

6) % INT EXP

Integer expected.

7) / LEV ERROR

Level is negative or greater than 31.

8) / $ EXP

Dollar expected.

9) / IDEN EXP

Identifier expected.

10) / L-EXP

Letter 'L' expected.

11) / R-EXP

Letter 'R' expected.

12) / A-EXP

Letter 'A' expected.

13) / X/Y EXP

Letter 'X' or 'Y' expected.

14) / S/R EXP

Letter 'S' or 'R' expected.

15) RANGE OUT

Cursor out of display range and cannot be positioned.

REFERENCES

[1] Besant, C.B., 'Computer Aided Design for Engineers', Ellis Horwood Ltd., New York, 1981.

[2] William M. Newman, Robert F. Sproull, 'Principles of Interactive Computer Graphics', Second Edition, International Students' Edition, McGraw-Hill, Kogakusha.

[3] C. Cavagna and V. Cugini, 'Data Structure for the Description and Handling of Engineering Drawings', Computer Aided Design, vol. 9, No. 1, Jan.1977, pp 17-22.

[4] Robin Williams, 'A Survey of Data Structures for Computer Graphics Systems', Computing Surveys, vol. 3, No.1, March 1971, pp 1-21.

[5] Scott, E., 'Introduction to Interactive Computer Graphics', John Wiley, New York, 1981.

[6] Machover, 'Display Systems: Computer Graphics', the Optical Publishing Co., Pittsfield, MA, 1979.

[7] Tektronix 4006-1, Computer Display Terminal User's Manual.

[8] Tektronix 4662, Interactive Plotter User's Manual.

[9] Monitor Calls Manual, DEC-10 Software Notebooks.

[10] 'Indian Standard Code Practice for General Engineering Drawings', Indian Standards Institution, Second Edition, February, 1976.

DRAFTING PACKAGE FOR ENGINEERING DRAWINGS

DESIGNED AND IMPLEMENTED AT
INDIAN INSTITUTE OF TECHNOLOGY - KANPUR

BY

C.H. VENKATESH MURTHY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

MASTER OF TECHNOLOGY

VERSION 0.0 DATE: 30-JUN-82

```
PROGRAM  DRAFT(AINF,AINHRK);
  const
        GS=29; %  ENTER GRAPHICS MODE  \
        US=31; %  LEAVE    "       "    \
        ESC=27; % ESCAPE CHARACTER    \
        FF=12;  % FORM FEED   \
        BEL=7;
        CR=13;  % CARRIAGE RETURN \
        MAXNLS= 35;  % MAX NO OF LINES IN COMMAND AREA  \
        MAXNCS= 13;    % MAX NO OF CHARS   "    "     \
        HCHARWID=17; % HORIZONTAL CHAR WIDTH IN COMMAND AREA  \
        VCHARWID=20; % VERTICAL       "    "      "    "     \
        CHRORGX=780; % CHARACTER ORIGIN IN COMMAND AREA  \
        CHRORGY=710; %     "       "    "     "     "    \
        MAXX=750; MAXY=750; MINX=0;MINY=0; % GRAPHICS AREA  \
        LLEMAX=74; % MAX. INPUT LINE LENGTH  \
        DLLEN= 0.3; % LENGTH OF DOTTED LINE SEGMENT IN cms  \
        CHLEN= 1.5; % LENGTH OF CHAIN    "     "    "    \
        ARROWLEN= 0.6; % LENGTH OF ARROW IN cms  \
        ARCERR= 0.01; % MAX. ERROR ALLOWED IN CIRCLE APPROXIMATION \
        PSWSIZE=1.2; % POINT SEARCH WINDOW USED IN FIND COMMAND  \
        SLDMAX=1.4; %MAX SEARCH DISTANCE FOR LINES  \
        %------------------------------------------------------------  \
        IDSIZE=5;  % LENGTH OF IDENTIFIER  \
        NMCC= 35;  % NO OF MULTIPLE CHAR COMMANDS  \
        NSCC=13;     % NO OF SINGLE    "      "     \
        NMONC=10; % NO. OF MONITOR COMMANDS  \
        NIMAX=8; % MAX. NO. OF SIGNIFICANT DIGITS IN A NUMBER \
        NDMAX=5; % MAX. NO. OF DECIMAL PLACES      "      "   \
        MAXOPCODE= 63;
        MAXLEVEL = 31;
        MAXCODE= 3000; % MAXIMUM CODE AREA  \
        ERRMAX=14; % MAX. NO. OF ERRORS \
        %------------------------------------------------------------  \
  type
        WCOORDINATE = record
                          X,Y:real
                      end;
        SCOORDINATE = record
                          X,Y:integer
                      end;
        BOX = record
                  XMIN,XMAX,YMIN,YMAX:real
              end;
        IDENTIFIER = packed array[1..IDSIZE] of char;
        SYMBOLS= (REALCON,INTCON,IDEN,PLUS,MINUS,CRTN,DOLLAR,NUL);
        %------------------------------------------------------  \
        %  DATA STRUCTURE  \
        INSTTYP=(LINEI,ARCI,CARCI,STGI,CSTGI,MACT,MACENDI);
        CARCTYP= record
                     RADIUS,ANGLE:real
                 end;
        CSTGTYP= packed array[1..10] of char;
```

```
INSTRUCTIOn= packed record
                          U,LINBEGIN: boolean;
                          OPCODE:0..MAXOPCODE;
                          LEV: 0..MAXLEVEL;
                          case INSITYP of
                               LINET:(LTN: WCOORDINATe);
                               ARCI: (CENTRE:WCOORDINATe);
                               CARCI:(ARC:CARCIYP);
                               STGI: (ST:WCOORDINATe);
                               CSTGI:(STC:CSTGIYP);
                               MACI: (MINDIM:WCOORDINATe);
                               MACENDI:(MAXDIM:WCOORDINATe)

                    end;
%--------------------------------------------------- \
MODETYP= (LINEM,SYMM,MACM,DEVM);
MMTYP=(MAIN,MACRO);
STATTYP=(NEW1,OLD);
%---------------------------------------------------- \
%  INPUT OUTPUT FILE TYPES   \
FILREC=(DIRWORD,CODEWORD);
DIRFIELD=(PTRS,DSZ);

DIRTYP=record
             NAME:IDENTIFIER;
             case DIRFIELD of
                  PTRS:(STRTP,ENDP:integer);
                  DSZ: (DSIZE,Z:integer)
        end;
FILECOMP=record
              case FILREC of
                   DIRWORD:(FDIR:DIRTYP);
                   CODEWORD:(FCODE:INSTRUCTIOn)
         end;

FILENAME=packed array[1..9] of char;
FILETYP= file of FILECOMP;
%---------------------------------------------- \
% HATCHING EDGE LIST TYPES   \
PNODE= ^EDGEREC;

EDGEREC = record
              P0,P1:WCOORDINATe;
              NEXT:PNODE
          end;
%---------------------------------------------- \

var
WINDOW:BOX;
WCURSOR,WFIX:WCOORDINATe; % CURSOR POSITION IN WORLD COORDINATE \
A,B,C,D:real; %TRANSFORMATION CONSTANTS  \
CURP,FIXP: WCOORDINATe;
%------------------------------------------------- \
```

```
VXMIN,VXMAX,VYMIN,VYMAX:integer; % VIEW PORT DIMENSIONS   \
OLDDVST:SCOORDINATE; % LAST SCREEN COORDINATES TRANSMITTED  \
SCURSTEP:real;
LCURSTEP:real;  % LARGE CURSER STEP  \
CCP,CLP:integer; % COMMAND CHARACTER & LINE POINTERS   \
%--------------------------------------------------------- \
MCCOMTAB: array [0..MMCC] of IDENTIFIER;
SCCOMTAB: packed array[0..MSCC] of char;
MUNCOMTAB:array [0..MMOUC] of IDENTIFIER;
ERRMESG: array[1..ERRMAX] of IDENTIFIER;

CH:char;
DIGITS,ALPHAS,DELIMIT:set of char;
ID:IDENTIFIER; SY:SYMBOLS;
RNUM: real; % REAL NUMBER OBTAINED AS SYMBOL  \
INUM:integer; % INTEGER OBTAINED AS SYMBOL   \
NUMFLAG:boolean;% true if numbeg exists   \
LINBUF: packed array [1..75] of char;% LINE BUFFER,SIZE=LLEMAX+1 \
CC,LL:integer;
SCALE:real;
HANGLE:real;  % HATCH ANGLE   \
SCREENHEAD:packed array[1..10] of char;

%--------------------------------------------------------- \
CODE: packed array[0..MAXCODE] of INSTRUCTION;
STARTCODE, ENDCODE:0..MAXCODE;
LEVEL:0.. MAXLEVEL;
PC:integer;
LINEMODE:boolean;% TO INDICATE DISCONTINUITY IN LINES   \
MINMAXDIM:BOX; % BOUNDING BOX DIMENSION OF SYMBOLS & MACROS \
LOADDIMENSION:boolean; % ENABLES BOUNDING BOX UPDATE   \
VISIBLE:boolean; % VISIBILITY OF LINE AFTER CLIPPING   \
%------------------------------------------------------------- \
DELETED: record
              DPC:integer;
              case MODE:MODETYP of
                  LINEM:(CASEND:integer);
                  SYMM:(CIRCLE:boolean);
                  MACM:(LSTPC:integer);
                  LEVM:(LEVO:integer)
         end;
LEVDISPLAY:boolean; % ENABLES LEVEL DISPLAY  \
DLEV:integer; % LEVEL # TO BE DISPLAYED   \
%------------------------------------------------- \
MAINF,MAINBAK:FILETYP;
FNMAIN,FNMAC,FNMBAK,FNMACBAK:FILENAME;
DIX:integer; % DIRECTORY INDEX TO FOUND PIC \
MACORIGIN: WCOORDINATE;
%----------------------------------------------- \
DWGSTATUS: packed record
                    DNAME:IDENTIFIER;
                    DTYP: MMTYP;
                    STATUS:STATTYP
```

```
                    end;
    MACFILFLAG:boolean;%TRUE IF MAC LIB EXISTS \
    DRAFTOVER: boolean;
    SAVEPIC:boolean;% TRUE IF DRAWING IS TO BE SAVED \
    DIRFOUND:boolean;% TRUE IF NAME IN DIR FOUND \
    MONITORMODE:boolean;% TRUE IF PROG IN MONITOR MODE \
    HATCHFLAG:boolean; % TRUE IF HATCHING IS ON   \
    %------------------------------------------------- \
    LISTHEAD, LISTTAIL, NEWLIST: PNODE;
    %------------------------------------------------- \

initprocedure;
    begin
        VXMIN:=MINX; VXMAX:=MAXX;
        VYMIN:=MINY; VYMAX:=MAXY
    end;

initprocedure; % INITIALISE COMMAND TABLE  \
    begin
        MCCOMTAB[0]:='     '; MCCOMTAB[1]:='LS   ';
        MCCOMTAB[2]:='LD   '; MCCOMTAB[3]:='LCH  ';
        MCCOMTAB[4]:='LA   '; MCCOMTAB[5]:='AL   ';
        MCCOMTAB[6]:='ALA  '; MCCOMTAB[7]:='CS   ';
        MCCOMTAB[8]:='CD   '; MCCOMTAB[9]:='CCH  ';
        MCCOMTAB[10]:='ARS  '; MCCOMTAB[11]:='ARD  ';
        MCCOMTAB[12]:='ARCH '; MCCOMTAB[13]:='ARA  ';
        MCCOMTAB[14]:='AAR  '; MCCOMTAB[15]:='AARA ';
        MCCOMTAB[16]:='PC   '; MCCOMTAB[17]:='PCR  ';
        MCCOMTAB[18]:='STEPC'; MCCOMTAB[19]:='CLEV ';
        MCCOMTAB[20]:='SLINE'; MCCOMTAB[21]:='SSYM ';
        MCCOMTAB[22]:='SLEV '; MCCOMTAB[23]:='DLEV ';
        MCCOMTAB[24]:='ER   '; MCCOMTAB[25]:='EQ   ';
        MCCOMTAB[26]:='HATCH'; MCCOMTAB[27]:='CMOV ';
        MCCOMTAB[28]:='SWIND'; MCCOMTAB[29]:='DWIND';
        MCCOMTAB[30]:='CALL '; MCCOMTAB[31]:='STG  ';
        MCCOMTAB[32]:='HANGL'; MCCOMTAB[33]:='ORIGN';
        MCCOMTAB[34]:='SMAC '; MCCOMTAB[35]:='SSTG ';
        SCCOMTAB:= ' NM,.^]<>QX?FR'
    end;

initprocedure; % INITIALISE MONITOR COMMAND TAB \
    begin

        MONCOMTAB[0]:='     '; MONCOMTAB[1]:='TYPE ';
        MONCOMTAB[2]:='TYPEM'; MONCOMTAB[3]:='SCALE';
        MONCOMTAB[4]:='DIR  '; MONCOMTAB[5]:='DIRM ';
        MONCOMTAB[6]:='DRAFT'; MONCOMTAB[7]:='MACRO';
        MONCOMTAB[8]:='DEL  '; MONCOMTAB[9]:='DELM ';
        MONCOMTAB[10]:='KJOB '
    end;

initprocedure; % INITIALISE ERROR MESSAGE \
    begin
```

```
            ERRMESG[1]:='CMDER';  ERRMESG[2]:='L-VE?';
            ERRMESG[3]:='R-VE?';  ERRMESG[4]:='NUMEX';
            ERRMESG[5]:='INTEX';  ERRMESG[6]:='LEVBG';
            ERRMESG[7]:='S EXP';  ERRMESG[8]:='IDEXP';
            ERRMESG[9]:='L-EXP';  ERRMESG[10]:='A-EXP';
            ERRMESG[11]:='X/YEX'; ERRMESG[12]:='RANGE';
            ERRMESG[13]:='R-EXP'; ERRMESG[14]:='S/REX';

        end;


procedure  OUTCHR(C:integer);
    extern;
procedure  INCHR(var C:char);
    extern;
procedure  IONEQU(T:integer);
    extern;
procedure  NOECHO;
    extern;
procedure WAIT(T:integer);
    extern;
function SKPINC:boolean;
    extern;

procedure  SETWVCONSTant;
        begin
            with  WINDOW do
              begin
                  A:=(VXMAX-VXMIN)/(XMAX-XMIN);
                  B:= VXMIN - A*XMIN;
                  C:=(VYMAX-VYMIN)/(YMAX-YMIN);
                  D:=VYMIN - C*YMIN
              end
        end;

procedure  SETWINDOW(X1,X2,Y1,Y2:real);
        begin
            with WINDOW do
              begin
                  XMIN:=X1; XMAX:=X2;
                  YMIN:=Y1; YMAX:=Y2
              end;
            SETWVCONSTant
        end;
      % SETWINDOW  \

procedure  SETCURSORStep;
        begin
            with WINDOW do
              begin
                  LCURSTEP:=(XMAX-XMIN)/20;
                  SCURSTEP:= LCURSTEP/20
```

```
            end
        end;
      % SETCURSORSTEP \

procedure INITIALISE;
        begin
            DIGITS:=['0'..'9'];
            ALPHAS:=['A'..'Z'];
            DELIMIT:=[' ',';',','];
            LINEMODE:=false;
            LEVDISPLAY:=false;
            HATCHFLAG:=false;
            HANGLE:=45.0;
            SCALE:=1.0;
            LOADDIMENSion:=false; LEVEL:=1;
            PC:=1; STARTCODE:=1;ENDCODE:=0;
            with WINDOW do
              begin  WFIX.X:=XMIN;WFIX.Y:=YMIN;
                   WCURSOR.X:=XMIN; WCURSOR.Y:=YMIN
              end
        end; % INITIALISE \
      %-------------------------------------------------------- \
      % THIS MODULE IS DEVICE DEPENDENT. THIS IS WRITTEN FOR THE      \
      % TEKTRONICS GRAPHICS TERMINAL 4006-1    \
procedure CLEARSCREEN;
        begin
            OUTCHR(ESC);
            IONEOU(FF);
            WAIT(1)
        end;

procedure  SENDCOORDInate(P:SCOORDINATE);

        var
            YH1,YH2,YL1,YL2,XH1,XH2,XL:integer;
        begin
            YH1:=OLDDVST.Y div 32; YH2:=P.Y div 32;
            YL1:=OLDDVST.Y mod 32; YL2:=P.Y mod 32;
            XH1:=OLDDVST.X div 32; XH2:=P.X div 32;
            XL:= P.X mod 32;
            if YH2 <> YH1  then OUTCHR(32+YH2);
            if (YL2 <> YL1)or (XH2 <>XH1) then
                  OUTCHR(96+YL2);
            if (XH2 <> XH1) then OUTCHR(32+XH2);
            OUTCHR(64+XL); OLDDVST:=P
        end;

procedure  DRAWLINE(P1,P2:SCOORDINATE);
        begin
            OUTCHR(GS); SENDCOORDInate(P1);
            SENDCOORDInate(P2); OUTCHR(US)
        end;
```

```
procedure MOVETO(P:SCOORDINATe);
      begin
          OUTCHR(GS);SENDCOORDInate(P);
          OUTCHR(US)
      end;

procedure CURSORHOME; % SET TO 0,780 \

      begin
          OLDOVST.X:=0; OLDOVST.Y:=780;
          MOVETO(OLDOVST)
      end;

    %------------------------------------------------------------ \
procedure  WORLDTOSCReen(W:WCOORDINATe; var P:SCOORDINATe);
      begin
          P.X:=TRUNC(A*W.X+B);
          P.Y:=TRUNC(C*W.Y+D)
      end;

procedure  MOVE;
      var
          P:SCOORDINATe;
      begin
          WORLDTOSCReen(WCURSOR,P);
          MOVETO(P)
      end;

function  INSIDE(W:BOX; P:WCOORDINATe):boolean;
      begin
          INSIDE:=true;
          with  W,P do
                if (X<XMIN) or (X>XMAX) or(Y<YMIN) or (Y>YMAX)
                       then    INSIDE:=false
      end;

procedure UPDATEDIMEnsion;
      var
          TEMP:WCOORDINATe; I:integer;
      begin
          TEMP:=FIXP;
          for I:=1 to 2 do
            begin
                if not (INSIDE(MINMAXDIM,TEMP)) then
                  begin
                      with MINMAXDIM,TEMP do
                        begin
                            if X < XMIN then
                                  XMIN:=X
                            else
                                  if X > XMAX then
                                         XMAX:=X;
                            if Y < YMIN then
                                  YMIN:=Y
```

```
                              else
                                    if Y > YMAX then
                                          YMAX:=Y
                        end;
                      % WITH   \
                   end;
                 TEMP:=CURP
              end;
            % FOR   \
         end;
       % UPDATEDIMENSION   \

procedure   ENTERLIST(P0,P1:WCOORDINATE);
         var
              T:PNODE;

         begin
              NEW(T); T^.P0:=P0;
              T^.P1:=P1; T^.NEXT:=nil;
              if LISTTAIL =nil then
                begin
                     LISTHEAD:=T; LISTTAIL:=T
                end
              else
                begin
                     LISTTAIL^.NEXT:=T;
                     LISTTAIL:=T
                end
         end;
       % ENTERLIST   \

procedure   CLIPWINDOW;
         label
              99;
              type
              EDGE=(LEFT,RIGHT,BOTTOM,TOP);
              TESTCODE=set of EDGE;
         var
              C,C1,C2:TESTCODE;
              X,Y,X1,Y1,X2,Y2:real;

         procedure   CODE(X,Y:real; var C:TESTCODE);
               begin
                    C:=[];
                    with WINDOW do
                      begin
                           if X<XMIN  then C:=[LEFT]
                           else
                                 if X>XMAX then C:=[RIGHT];
                           if Y<YMIN then C:=C+[BOTTOM]
                           else
                                 if Y>YMAX then C:=C+[TOP]
                      end
```

```
      end;

begin % CLIPWINDOW \
    VISIBLE:=true;X1:=FIXP.X;Y1:=FIXP.Y;
    X2:=CURP.X;Y2:=CURP.Y;
    CODE(X1,Y1,C1);CODE(X2,Y2,C2);
    while  (C1<>[]) or (C2<>[]) do
      begin
          if (C1*C2)<>[] then
            begin
                VISIBLE:=false; goto 99
            end;
          C:=C1;
          if C=[] then C:=C2;
          with  WINDOW do
            begin
                if LEFT in C then
                  begin  % CROSSES LEFT EDGE \
                      Y:=Y1+(Y2-Y1)*(XMIN-X1)/(X2-X1);
                      X:=XMIN
                  end
                else
                      if RIGHT in C then
                        begin  % CROSSES RIGHT EDGE \
                            Y:=Y1+(Y2-Y1)*(XMAX-X1)/(X2-X1);
                            X:=XMAX
                        end
                      else
                            if BOTTOM in C then
                              begin % CROSSES BOTTOM EDGE  \
                                  X:=X1+(X2-X1)*(YMIN-Y1)/(Y2-Y1);
                                  Y:=YMIN
                              end
                            else
                                  if TOP in C then
                                    begin % CROSSES TOP EDGE  \
                                        X:=X1+(X2-X1)*(YMAX-Y1)/(Y2-Y1);
                                        Y:=YMAX
                                    end
            end ;
          %WITH  \
          if C=C1 then
            begin
                X1:=X;Y1:=Y;
                CODE(X1,Y1,C1)
            end
          else
            begin
                X2:=X;Y2:=Y;
                CODE(X2,Y2,C2)
            end
      end ;
    % WHILE  \
    WFIX.X:=X1; WFIX.Y:=Y1;
```

```
            WCURSOR.X:=X2;WCURSOR.Y:=Y2;
            99:
        end;
     % CUTPWINDOW \

procedure  DRAW;
        var
            P1,P2:SCOORDINATE;
        begin  % DRAW  \
            if HATCHFLAG then
                   ENTERLIST(FIXP,CURP)
            else
              begin
                   if LOADDIMENSION then
                          UPDATEDIMENSION;
                   CUTPWINDOW;
                   if VISIBLE then

                     begin
                          WORLDTOSCREEN(WCURSOR,P2);
                          WORLDTOSCREEN(WFIX,P1);
                          DRAWLINE(P1,P2);
                          WFIX:=WCURSOR
                     end
              end
        end;
     %DRAW  \

procedure  INITSCREEN;
        var
            P1,P2:SCOORDINATE;

        procedure WRITESCREENhead;
              var
                   I:integer;
                   P:SCOORDINATE;
              begin
                   for I:=6 to 10 do
                          SCREENHEAD[I]:= DWGSTATUS.DNAME[I-5];
                   P.X:=780;P.Y:=MAXY-19; MOVETO(P);
                   WRITE(TTY,SCREENHEAD);BREAK
              end;

        procedure  DISPLAYWINDOW;
              var
                   P:SCOORDINATE;
              begin
                   P.X:=MINX;P.Y:=755;
                   MOVETO(P);
                   with WINDOW do
                          WRITE(TTY,'WINDOW DIM:',XMIN:13:5,',',YMIN:13:5,',',XMAX:13:5,',',YMAX:13:5);
                   BREAK
              end;
```

```
    begin
        CLEARSCREEN; CURSURHOME;
        P1.X:=MINX; P1.Y:=MINY; P2.X:=1023; P2.Y:=MINY;
        DRAWLINE(P1,P2); P1:=P2; P2.Y:=MAXY;
        DRAWLINE(P1,P2); P1:=P2; P2.X:=MINX;
        DRAWLINE(P1,P2); P1:=P2; P2.Y:=MINY;
        DRAWLINE(P1,P2); P1.X:=MAXX; P2.X:=MAXX;
        DRAWLINE(P1,P2); P1.Y:=MAXY-20; P2.X:=1023;
        P2.Y:=MAXY-20; DRAWLINE(P1,P2);
        CCP:=0; CLP:=0; DISPLAYWINDOW;
        WRITESCREENhead; MOVE
    end;
  % INITSCREEN  \

procedure SELFERASE;
  forward;


procedure  WRITECH(C:char);
    var
        P:SCOORDINATE;
    begin
        if not MONITORMODE then
          begin
             if (CLP=MAXWLS) and (CCP=MAXWCS) then  SELFERASE;
             if (CCP=MAXWCS) then
               begin
                  CLP:=CLP+1; CCP:=0
               end;
             P.X:=CHRORGX+CCP*HCHARWID;
             P.Y:=CHRORGY-CLP*VCHARWID; CCP:=CCP+1;
             MOVETO(P); OUTCHR(ORD(C)); MOVE
          end
        else OUTCHR(ORD(C))
    end;

procedure ERROR(N: integer);
    var
        I:integer;

    begin
        OUTCHR(BEL);
        if N<>0 then
             for I:=1 to IDSIZE do
                  WRITECH(ERRMESG[N,I])

    end;
procedure  GETCH;
    begin
        CC:=CC+1; CH:=LINBUF[CC]
    end;

procedure  GETSYM;
```

```
var
    I,D:integer;
begin
    while CH in DELIMIT do GETCH;
    if CH in ALPHAS  then
      begin %IDENTIFIER  \
          I:=0; ID:='      ';
          repeat
                if I<IDSIZE  then
                  begin
                      I:=I+1; ID[I]:=CH
                  end;
                GETCH
          until  not (CH in ALPHAS);
          SY:=IDEN
      end
    else
          if CH in (DIGITS+ ['.']) then
            begin % REAL OR INTEGER NO. \
                INUM:=0;I:=0;
                if CH in DIGITS  then
                  begin
                      SY:=INTCON;
                      repeat
                            INUM:=INUM*10+ORD(CH)-ORD('0');
                            I:=I+1;GETCH
                      until not (CH in DIGITS);
                      if I>NIMAX then
                        begin
                            ERROR(0); SY:=NUL;INUM:=0
                        end
                  end;
                if CH='.'  then
                  begin  RNUM:=INUM; SY:=REALCON;
                      I:=0;D:=10;GETCH;
                      while CH in DIGITS do
                        begin
                            if I<NDMAX  then
                              begin
                                  RNUM:= RNUM + (ORD(CH)-ORD('0'))/D;
                                  I:=I+1;D:=D*10
                              end;
                            GETCH
                        end %WHILE \
                  end
            end
          else
                if (CH= CHR(CR)) then  SY:=CRTN
                else
                case  CH of
                      '-':
                            begin  SY:=MINUS; GETCH
                            end;
                      '+':
```

```
                                    begin SY:=PLUS;GETCH
                                    end;
                          '$':
                                    begin SY:=DOLLAR; GETCH
                                    end;
                          others:SY:=NUL
                      end%CASE \
        end;
    %GETSY% \

procedure MOVECURSOR(I:integer);
        begin
           with MCURSOR,WINDOW do
           case I of
                 1,5: % MOVE UP \
                        begin
                           if I=1 then   %LARGE STEP UP \
                                 Y:=Y+LCURSTEP
                           else  % SMALL STEP UP \
                                 Y:=Y+SCURSTEP;
                           if Y>YMAX  then Y:=YMIN+Y-YMAX;
                           MOVE
                        end;
                 2,6: %MOVE DOWN  \
                        begin
                           if I=2 then  %LARGE STEP DOWN \
                                 Y:=Y-LCURSTEP
                           else %SMALL STEP DOWN \
                                 Y:=Y-SCURSTEP;
                           if Y<YMIN then Y:=YMAX-YMIN+Y;
                           MOVE
                        end;
                 3,7:%MOVE LEFT \
                        begin
                           if I=3 then X:=X-LCURSTEP
                           else X:=X-SCURSTEP;
                           if X<XMIN  then  X:=XMAX-XMIN+X;
                           MOVE
                        end;
                 4,8: % MOVE RIGHT \
                        begin
                           if I=4 then X:=X+LCURSTEP
                           else
                                 X:=X+SCURSTEP;
                           if X>XMAX  then X:=XMIN+X-XMAX;
                           MOVE
                        end
           end%CASE \
        end;
    %MOVECURSOR \

procedure NEXTINSTRUCTION;
    forward;
```

```
procedure INTERPRET;
      label
          11;
          const    RADTODEG =57.29578;
      type
          ANGLES = record
                        C,S:real
                    end;
      var
          C:WCOORDINATE; R,A:real;
          P0,P1:WCOORDINATE;
          DOTTED:boolean;
      function  TRIVIALREJect(P0,P1:WCOORDINATE):boolean;
            type
                EDGE=(LEFT,RIGHT,BOTTOM,TOP);
                TESTCODE=set of EDGE;
            var
                C1,C2:TESTCODE;

            procedure  CODE(X,Y:real; var C:TESTCODE);
                  begin
                      C:=[];
                      with WINDOW do
                        begin
                            if X<XMIN  then C:=[LEFT]
                            else
                                  if X>XMAX then C:=[RIGHT];
                            if Y<YMIN then C:=C+[BOTTOM]
                            else
                                  if Y>YMAX then C:=C+[TOP]
                        end
                  end;
            begin % TRIVIALREJECT  \
                TRIVIALREJect:=false;
                CODE(P0.X,P0.Y,C1);
                CODE(P1.X,P1.Y,C2);
                if (C1*C2) <> [] then
                      TRIVIALREJect:=true
            end;
          %TRIVIALREJECT  \

      procedure LINEROUTINE(CTP:integer);
            var
                SINV,COSV,XDIF,YDIF,LENGTH,SEGLENGTH,SPACE:real;
                CUR1,SCUR,OFFSET:WCOORDINATE;
                I:integer;
            procedure TRANSFORM(var P:WCOORDINATE);
                  var
                      P1:WCOORDINATE;
                  begin
                      P1.X:=(P.X*COSV-P.Y*SINV)+OFFSET.X;
                      P1.Y:=(P.X*SINV+P.Y*COSV)+OFFSET.Y;
                      P:=P1
                  end;
```

```
procedure   DRAWARROW(I:integer);
      var
            ARROWH,ARROWX:real;
            SAVCUR,SAVFIX:WCOORDINATE;
      begin   SAVCUR:=CURP;SAVFIX:=FIXP;
            ARROWH:=SCALE*ARROWLEN/6;
            if I=0 then   ARROWX:=SCALE*ARROWLEN
            else
              begin
                  ARROWX:=LENGTH-(SCALE*ARROWLEN);
                  FIXP:=CURP
              end;
            CURP.X:=ARROWX; CURP.Y:=ARROWH;
            TRANSFORM(CURP);DRAW; CURP.X:=ARROWX;
            CURP.Y:=-ARROWH;TRANSFORM(CURP); DRAW;
            CURP:=SAVCUR;FIXP:=SAVFIX
      end;
    %DRAWARROW  \
begin %LINEROUTINE \
    OFFSET:=FIXP;XDIF:=CURP.X-FIXP.X;
    YDIF:=CURP.Y-FIXP.Y;LENGTH:=SQRT(XDIF*XDIF+YDIF*YDIF);
    if LENGTH <> 0 then
      begin SINV:=YDIF/LENGTH;
            COSV:=XDIF/LENGTH
      end
    else
      begin SINV:=0;
            COSV:=0
      end;
    case CTP of
          2:
                    begin % DOTTED LINE  \
                        SEGLENGTH:=SCALE*DLLEN; I:=TRUNC(LENGTH/SEGLENGTH);
                        CUR1.X:=0;CUR1.Y:=0;SCUR:=CURP;
                        while I>= 2  do
                          begin    CUR1.X:=CUR1.X+SEGLENGTH;CURP:=CUR1;
                              TRANSFORM(CURP); DRAW; CUR1.X:=CUR1.X+SEGLENGTH;
                              FIXP:=CUR1;TRANSFORM(FIXP); I:=I-2
                          end;
                        %WHILE \
                        if I=1 then
                          begin CURP:=SCUR;DRAW
                          end
                        else
                              if CUR1.X=LENGTH then
                                begin
                                    CUR1.X:=CUR1.X-SEGLENGTH/2;
                                    FIXP:=CUR1;CURP:=SCUR;TRANSFORM(FIXP);
                                    DRAW
                                end
                              else
                                begin  CURP:=SCUR; DRAW
                                end
```

```
                    end;
        3:          begin %CHAIN LINE \
                        SEGLENGTH:=SCALE*(CHLEN +CHLEN/5+0.4);
                        I:=TRUNC(LENGTH/SEGLENGTH);CUR1.X:=0;CUR1.Y:=0;
                        SCUR:=CURP; SEGLENGTH:=SCALE*CHLEN; SPACE:=SCALE*0.2;
                        while  I>=1 do
                          begin
                              CUR1.X:=CUR1.X+SEGLENGTH;CURP:=CUR1;TRANSFORM(CURP);
                              DRAW;CUR1.X:=CUR1.X+SPACE; FIXP:=CUR1;TRANSFORM(FIXP);
                              CUR1.X:=CUR1.X+SEGLENGTH/5;CURP:=CUR1;TRANSFORM(CURP)
                              ;DRAW;CUR1.X:=CUR1.X+SPACE;FIXP:=CUR1;TRANSFORM(FIXP);I:=I-1
                          end;
                        %WHILE \
                        if  (CUR1.X+SEGLENGTH+SPACE) >=LENGTH then
                          begin
                              CURP:=SCUR;DRAW
                          end
                        else
                          begin    CUR1.X:=CUR1.X+SEGLENGTH;CURP:=CUR1;TRANSFORM(CURP);
                              DRAW;CUR1.X:=CUR1.X+SPACE;FIXP:=CUR1;TRANSFORM(FIXP)
                              ;CURP:=SCUR; DRAW
                          end
                    end ;
                  %CHAINLINE  \
        4,5,6:      begin % ARROW LINE  \
                        if (CTP=5)or(CTP=6) then DRAWARROW(0);
                        if (CTP=4)or(CTP=6) then DRAWARROW(1);
                        DRAW
                    end % ARROW LINE  \
             end %CASE \
          end;
        %LINEROUTINE \

procedure GETDTHETA(R:real; var DTHETA:real);
      var
          R1:real;
      begin  R1:=R/SCALE;
          if R1>ARCERR then
                DTHETA:=2*ARCCOS(1-ARCERR/R1)*RADTODEG
          else     DTHETA:=360
      end;
    %GETDTHETA \

procedure TRANS(CENTRE:WCOORDINATE;A:ANGLES;var T:WCOORDINATE);
      var
          P1:WCOORDINATE;
      begin  T.X:=T.X-CENTRE.X;
          T.Y:=T.Y-CENTRE.Y;
          with  CENTRE,A do
            begin   P1.X:=X+T.X*C-T.Y*S;
                P1.Y:=Y+T.X*S+T.Y*C
            end;
```

```
            F:=0;
        end;
      % DRAWS \

procedure   DRAWARC(CENTRE:WCOORDINATE;A:ANGLES; N:integer);
        var
            I:integer;
        begin  for I:=1 to N do
               begin    FIXP:=CURP;
                   TRANS(CENTRE,A,CURP);
                   if DOTTED  then
                     begin
                         if (I mod 2) <> 0 then DRAW
                     end
                   else  DRAW
               end
        end;
      % DRAWARC  \

procedure   CALANGLE(DTHETA:real; var A:ANGLES);
        begin
            A.C:=COSD(DTHETA);
            A.S:=SIND(DTHETA)
        end;

procedure   GETSEGNO(THETA:real; var DTHETA:real; var N:integer);
        begin
            if THETA<0 then N:=TRUNC(ABS(THETA/DTHETA)+0.5)
            else
                  N:=TRUNC(THETA/DTHETA+0.5);
            DTHETA:=THETA/N
        end;

procedure     DRAWDOTTEDarc(THETA,DTHETA:real;var N1:integer;R:real;
                                   A:ANGLES;N:integer;CENTRE:WCOORDINATE);
        var
            A1:ANGLES; DTHETA1:real;
            I:integer; SAVCUR:WCOORDINATE;
        begin
            DTHETA1:= DLLEN*SCALE*RADTODEG/R;
            GETSEGNO(THETA,DTHETA1,N1);
            CALANGLE(DTHETA1,A1);
            if  ABS(DTHETA1)<= ABS(DTHETA ) then
              begin  DOTTED:=true;
                  DRAWARC(CENTRE,A1,N1)
              end
            else
              begin
                  GETSEGNO(DTHETA1,DTHETA,N);
                  CALANGLE(DTHETA,A);
                  for I:=1 to N1 do
                    begin
                        FIXP:=CURP;
                        TRANS(CENTRE,A1,CURP);
```

```
                  if (I mod 2) <> 0 then
                    begin
                        SAVCUR:=CURP; CURP:=FIXP;
                        DRAWARC(CENTRE,A,N); CURP:=SAVCUR
                    end

               end
            end;
          % ELSE \
        end;
      %DRAWDIFFERARC  \

procedure   DRAWCHAINARC(THETA,DTHETA:real;R:real;CENTRE:WCOORDINATe) ;
      var
          A1,A2,A3,A4:ANGLES;
          DTHETA1,DTHETA2,DTHETA3,DTHETA4:real;
          I,J,K,N1: integer;  SEGLEN:real;
          SAVCUR:WCOORDINATe;
      begin
          SEGLEN:=CHLEN+ CHLEN/5+ 0.4;
          DTHETA1:=SEGLEN*SCALE*RADTODEG/R;
          GETSEGNO(THETA,DTHETA1,N1);
          CALANGLE(DTHETA1,A1);
          DTHETA3:=CHLEN*DTHETA1/(5*SEGLEN);
          DTHETA4:=0.2*DTHETA1/SEGLEN;
          DTHETA2:=DTHETA1-DTHETA3- 2*DTHETA4;
          CALANGLE(DTHETA4,A4);
          if ABS(DTHETA2 )<= ABS(DTHETA )then  J:=1
          else
            begin   J:=TRUNC(ABS(DTHETA2/DTHETA)+0.5);
                DTHETA2:=DTHETA2/J
            end;
          CALANGLE(DTHETA2,A2);
          if ABS(DTHETA3 )<= ABS(DTHETA )then  K:=1
          else
            begin   K:=TRUNC(ABS(DTHETA3/DTHETA)+0.5);
                DTHETA3:=DTHETA3/K
            end;
          CALANGLE(DTHETA3,A3);
          for I:=1 to N1 do
            begin
                SAVCUR:=CURP;
                DRAWARC(CENTRE,A2,J);
                TRANS(CENTRE,A4,CURP);
                DRAWARC(CENTRE,A3,K);
                FIXP:=CURP;
                CURP:=SAVCUR;
                TRANS(CENTRE,A1,CURP)
            end;
        end;
      %DRAWCHAINARC \

procedure ARCROUTINE(CENTRE:WCOORDINATe;THETA:real;R:real;CIP:integer);
```

```
var
    A:ANGLES;DTHETA:real;N:integer;
procedure  DOTTEDARC;
      var
          N1:integer;
      begin
          DRAWDOTTEDarc(THETA,DTHETA,N1,R,A,N,CENTRE);
          if (N1 mod 2)=0  then
            begin
                FIXP.X:=FIXP.X+(CURP.X-FIXP.X)/2;
                FIXP.Y:=FIXP.Y+(CURP.Y-FIXP.Y)/2;
                DRAW
            end
      end;
    % DOTTEDARC  \

procedure  CHAINARC;
      begin
          DRAWCHAINarc(THETA,DTHETA,R,CENTRE);
          DRAW
      end;
    %CHAINARC  \

procedure  ARROWARCS( CIP:integer);
      var
          SCUR:WCOORDINATe; T:integer;

      procedure  DRAWARROW(I:integer);
            var
                A:ANGLES;XDIF,YDIF,LENGTH:real;
                OFFSET:WCOORDINATe;

            procedure  TRANSFORM( var P:WCOORDINATe);
                  var
                      T:WCOORDINATe;
                  begin
                      with A do
                        begin
                            T.X:= P.X*C -P.Y*S + OFFSET.X;
                            T.Y:= P.X*S +P.Y*C + OFFSET.Y
                        end;
                      P:=T
                  end;
                % TRANSFORM  \

            procedure    ARROW(I:integer);
                  var
                      ARROWH,ARROWX:real;
                      SAVCUR,SAVFIX:WCOORDINATe;
                  begin    SAVCUR:=CURP;SAVFIX:=FIXP;
                      ARROWH:=SCALE*ARROWLEN/6;
                      if I=0 then    ARROWX:=SCALE*ARROWLEN
                      else
                        begin
```

```
                                ARROWX:=LENGTH-(SCALE*ARROWLEN);
                                FIXP:=CURP
                            end;
                         CURP.X:=ARROWX; CURP.Y:=ARROWH;
                         TRANSFORM(CURP);DRAW; CURP.X:=ARROWX;
                         CURP.Y:=-ARROWH;TRANSFORM(CURP); DRAW;
                         CURP:=SAVCUR;FIXP:=SAVFIX
                     end;
                   %ARROW   \
              begin % DRAWARROW   \
                   OFFSET:=FIXP;XDIF:=CURP.X-FIXP.X;
                   YDIF:=CURP.Y-FIXP.Y;LENGTH:=SQRT(XDIF*XDIF+YDIF*YDIF);
                   if LENGTH <> 0 then
                     begin A.S:=YDIF/LENGTH;
                         A.C:=XDIF/LENGTH
                     end
                   else
                     begin A.S:=0;
                         A.C:=0
                     end;
                   ARROW(T)
              end;
            %DRAWARROW   \
        begin  % ARROWARCS   \
            if (CTP=14) or (CTP=15) then
              begin
                   SCUR:=CURP; FIXP:=CURP;
                   if (R/SCALE)<2 then I:=2
                   else I:=1;
                   while I <> 0 do
                     begin
                         TRANS(CENTRE,A,CURP);
                         I:=I-1
                     end;
                   DRAWARROW(0); CURP:=SCUR
              end;
            DRAWARC(CENTRE,A,N);
            if (CTP=13) or (CTP=15) then
              begin
                   SCUR:=CURP; DRAWARROW(1);
                   CURP:=SCUR;

                   if INSIDE(WINDOW,CURP) then
                     begin
                         WCURSOR:=CURP;WFIX:=CURP;MOVE
                     end
              end
        end;
      %ARROWARCS   \
begin %ARCROUTINE   \
      GETDTHETA(R,DTHETA);
      GETSEGND(THETA,DTHETA,N);
      DOTTED:=false;
      CALANGLE(DTHETA,A);
```

```
        case CTP of
              10: DRAWARC(CENTRE,A,N);
              11:
                    if not HATCHFLAG then  DOTTEDARC;
              12:
                    if not HATCHFLAG then  CHAINARC;
              13,14,15:
                    if not HATCHFLAG then  ARROWARCS(CTP)
        end
    end;
  %ARCROUTINE  \

procedure  CIRCLEROUTine(CENTRE:WCOORDINATE;R:real;CTP:integer);
      var
         A:ANGLES; DTHETA:real;N:integer;

      procedure DOTTEDCIRCle;
            var
               N1:integer;
            begin
               DRAWDOTTEDarc(360,DTHETA,N1,R,A,N,CENTRE);

               if (N1 mod 2)=0 then
                     if INSIDE(WINDOW,CURP) then
                       begin
                           WCURSOR:=CURP;MOVE
                       end
            end;
         % DOTTEDCIRCLE  \
      procedure  CHAINCIRCle;
            begin
               DRAWCHAINArc(360,DTHETA,R,CENTRE);
               if INSIDE(WINDOW,CURP) then
                  begin
                     WCURSOR:=CURP; MOVE ;WFIX:=WCURSOR
                  end
            end;
         %CHAINCIRCLE \

      begin %CIRCLEROUTINE \
         GETDTHETA(R,DTHETA);
         GETSEGNO(360,DTHETA,N);
         CALANGLE(DTHETA,A); DOTTED:=false;
         CURP.X:=CENTRE.X+R; CURP.Y:=CENTRE.Y;
         case CTP of
               7: DRAWARC(CENTRE,A,N);
               8:
                    if not HATCHFLAG then  DOTTEDCIRCle;
               9:
                    if not HATCHFLAG then  CHAINCIRCLe
         end % CASE  \
      end;
    % CIRCLEROUTINE  \
```

```
procedure   WRITESTRING;
     var
          I,J,SAVPC,MAXLEN:integer;
          P0:SCOORDINATE; C:char;
          EOFSTRING:boolean;
     begin
          SAVPC:=PC;
          WORLDTOSCReen(WCURSUR,P0);
          MAXLEN:=MAXX-P0.X;
          J:= MAXLEN  div HCHARWID;
          EOFSTRING:=false; I:=0;PC:=PC+1;
          while (J>0) and not EOFSTRING  do
            begin
               if I=10  then
                 begin
                      PC:=PC+1; I:=0
                 end;
               I:=I+1; C:=CODE[PC].STC[I];
               if C='$' then EOFSTRING:=true
               else
                 begin
                      J:=J-1; MOVETO(P0);OUTCHR(ORD(C));
                      P0.X:=P0.X+HCHARWID
                 end
            end;
          % WHILE \
          PC:=SAVPC
     end;
   % WRITESTRING \

begin % INTERPRET  \
     while PC <= ENDCODE do
       begin
          with CODE[PC] do
            begin
               if LEVDISPLAY then
                    if LEV<> DLEV then goto 11;
               if not D then
               case OPCODE of
                    1,2,3,4,5,6:
                         begin %LINES \
                              if LINBEGIN then  FIXP:=LIN
                              else
                                begin
                                     CURP:= LIN;
                                     if OPCODE =1 then DRAW
                                     else
                                               if not HATCHFLAG then
                                                    LINEROUTINE(OPCODE);
                                     FIXP:=CURP
                                end;
                              %ELSE  \
                         end;
                        %LINES  \
```

```
                    7,8,9:
                          begin % CIRCLES  \
                              P0:= CODE[PC+2].MINDIM;
                              P1:=CODE[PC+3].MAXDIM;
                              if not TRIVIALREJect(P0,P1) then
                                begin
                                    C:=CENTRE; R:= CODE[PC+1].ARC.RADIUS;
                                    CIRCLEROUTine(C,R,OPCODE)
                                end
                          end;
                        %CIRCLES  \
                    10,11,12,13,14,15:
                          begin % ARCS  \
                              P0:=CODE[PC+3].MINDIM;
                              P1:=CODE[PC+4].MAXDIM;
                              if not TRIVIALREJect(P0,P1) then
                                begin
                                    C:=CENTRE;R:=CODE[PC+1].ARC.RADIUS;
                                    A:=CODE[PC+1].ARC.ANGLE;
                                    CURP:= CODE[PC+2].LIN;
                                    ARCROUTINE(C,A,R,OPCODE)
                                end
                          end ;
                        % ARCS \
                    31: % STRING \
                          begin
                              P0:=SF;
                              if INSIDE(WINDOW,P0) then
                                begin
                                    WCURSOR:=P0; MOVE;
                                    WRITESTRING
                                end
                          end  %STRING \
                end;
                % CASE  \
            end;
            % WITH  \
            11:
            NEXTINSTRUCtion
        end;
        % WHILE  \
    end;
    %INTERPRETER  \
procedure SELFERASE;
    var
        SCUR,SFIX,SWCUR,SWFIX:WCOORDINATe;
    begin
        SCUR:=CURP;SFIX:=FIXP;SWFIX:=WFIX;
        SWCUR:=WCURSOR; INITSCREEN;
        PC:=STARTCODE; INTERPRET;
        CURP:=SCUR;FIXP:=SFIX;WFIX:=SWFIX;
        WCURSOR:=SWCUR
    end;
```

```
% SELFERASE \

procedure  FILLBUFFER(var BUFFCOND:boolean);
      label
          1,2;
          var   RUBOUT:char;
      begin
          RUBOUT:=CHR(127); BUFFCOND:=true;

          LL:=1; LINBUF[LL]:=CH; WRITECH(CH);
          INCHRW(CH);
          1:
          while (CH <> CHR(CR)) and (CH <> RUBOUT) do
            begin
                if LL<LLMAX then
                  begin
                      LL:=LL+1; LINBUF[LL]:=CH;WRITECH(CH)
                  end;
                INCHRW(CH)
            end;
          %WHILE \
          if CH=RUBOUT then
            begin
                WRITECH('\');
                while CH=RUBOUT  do
                  begin
                      if LL>=1 then
                        begin
                            WRITECH(LINBUF[LL]); LL:=LL-1;
                            INCHRW(CH)
                        end
                      else
                        begin
                            WRITECH('\'); BUFFCOND:=false;
                            goto 2
                        end
                  end;
                % WHILE  \
                WRITECH('\'); goto 1
            end;
          %IF \
          LL:=LL+1; LINBUF[LL]:=CH; INCHRW(CH);
          2:
      end;
    %FILLBUFFER  \
procedure    GETNUMBER(var R:real);
      var
          SIGN:integer;
      begin
          SIGN:=1; NUMFLAG:=true; GETSYM;
          if SY=MINUS  then
            begin SIGN:=-1; GETSYM
            end;
```

```
            if SY=REALCON then  R:=RNUM*SIGN
            else
                 if SY=INTCON then R:=INUM*SIGN
                 else
                      NUMFLAG:=false
        end;

procedure  NEXTINSTRUCTION;
        begin
            with CODE[PC] do
            case OPCODE of
                 7,8,9: PC:= PC+4;
                 10,11,12,13,14,15: PC:=PC+5;
                 31:
                        begin
                            PC:=PC+1;
                            while (PC<=ENDCODE) and (CODE[PC].OPCODE=0) do
                                     PC:=PC+1
                        end;

                 50: PC:=PC+3;
                 others: PC:=PC+1
            end;
            %CASE  \
        end;
    % NEXTINSTRUCTION  \
procedure  GARCOLLECT;
        var
            TEMP:integer;

        begin
            PC:=STARTCODE; TEMP:=PC;
            CODE[ENDCODE+1].D:=true;
            while  PC <= ENDCODE  do
                 if  CODE[PC].D  then PC:=PC+1
                 else
                      while  not CODE[PC].D do
                        begin
                            CODE[TEMP]:=CODE[PC];
                            PC:=PC+1; TEMP:=TEMP+1
                        end;
            ENDCODE:= TEMP-1
        end;
    % GARCOLLECT \
procedure  GETWORD;
        begin
            if ENDCODE >= ((MAXCODE-20))  then GARCOLLECT;

            if ENDCODE <MAXCODE then
                 ENDCODE:=ENDCODE+1
            else
              begin ERROR(0); WRITELN;WRITELN(TTY,'CODE OVERFLOW')
              end
        end;
```

```
%SUBPROGRAMS  \

procedure SEARCHDIRECTORY(var F:FILETYP;ID:IDENTIFIER);
    forward;

procedure CHANGEMEMORY(var F:FILETYP; SP,RP:integer);
    forward;

procedure DRIVER;
    label
        111;
        var   CTP:integer; % COMMAND TABLE POINTER USED TO SEARCH COMMAND TABLE \
        BUFFUL:boolean; % TRUE IF LINE BUFFER CONTAINS SOME CHARS \
    procedure SCHCOMMAND;
        procedure PRINTNUMBER(N:real);
            var
                I,J:integer;
                INT:integer;%INTEGER PART OF THE GIVEN NUMBER \
                FRAC:real; % FRACTIONAL PART OF THE GIVEN NUMBER  \
                NUMBER: packed array[1..14] of char;
            procedure FILLNUMBER;
                var
                    TEMP:packed array[1..NTMAX] of char;
                    I:integer;
                begin
                    I:=0;
                    repeat
                          if I<NTMAX  then
                            begin
                                I:=I+1;
                                TEMP[I]:=CHR((INT mod 10 )+48)
                            end;
                          INT:= INT div 10;
                    until INT=0;
                    while I <> 0 do
                      begin
                          J:= J+1;
                          NUMBER[J]:= TEMP[I];
                          I:=I-1
                      end
                end;
                % FILLNUMBER  \
            begin % PRINTNUMBER  \
                INT:= TRUNC(N); FRAC:=N-INT;
                J:=0; FILLNUMBER;
                INT:=1;
                for I:=1 to NDMAX do INT:=INT*10;
                INT:=TRUNC(FRAC*INT);
                if INT >0 then
                  begin
                    J:=J+1;NUMBER[J]:='.';
                    % REMOVE TRAILING ZEROS  \
                    while (INT mod 10)=0  do
                          INT:=INT div 10;
```

```
                FILLNUMBER;
            end;
            for I:=1 to J do
                WRITECH(NUMBER[I])
        end;
      % OUTNUMBER  \
procedure DRAWFIX;
        var
            SCUR,SFIX:WCOORDINATE;
        begin
            SCUR:=CURP; SFIX:=FIXP;
            CURP:=WCURSOR; FIXP:=CURP;
            DRAW; CURP:=SCUR; FIXP:=SFIX
        end;
      % DRAWFIX  \
procedure  FIND;
        var
            SAVPC:integer;
            SEARCHWINDOW:BOX; FOUND:boolean;
            L:real; POINT:WCOORDINATE;
        begin
            L:=SCALE*PSWSIZE/2;
            with SEARCHWINDOW,WCURSOR do
              begin
                  XMIN:= X-L; XMAX:= X+L;
                  YMIN:=Y-L; YMAX:=Y+L
              end;
            FOUND:=false; SAVPC:=PC;PC:=STARTCODE;
            while (not FOUND)and (PC<=ENDCODE) do
              begin
                  if not CODE[PC].D then
                    begin
                        POINT:=CODE[PC].LIN;
                        if INSIDE(SEARCHWINDOW,POINT) then
                          begin
                              WCURSOR:= POINT; FOUND:=true;
                              MOVE
                          end
                    end ;
                  NEXTINSTRUCTION
              end;
            % WHILE  \
            if not FOUND then ERROR(0);
            PC:= SAVPC
        end;
      % FIND  \
procedure  REDISPLAY;
        var
            SCUR,SFIX,SWCUR,SWFIX:WCOORDINATE;

        begin
            SCUR:=CURP;SFIX:=FIXP;SWFIX:=WFIX;
            SWCUR:=WCURSOR;
            INITSCREEN;PC:=STARTCODE;
```

```
                INTERPRET;
                CURP:=SCUR;FIXP:=SFIX;WFIX:=SWFIX;
                WCURSOR:=SWCUR;MOVE;WRITECH('>')
            end;
          % REDISPLAY   \

      begin % SCHARCOMMAND   \
          case CTR of
                1,2,3,4,5,6,7,8: MOVECURSOR(CTR);
                9:   ;
                10:
                        begin WFIX:= WCURSOR; DRAWFIX
                        end;
                11:
                        begin % PRINT COORDINATES   \
                            WRITECH('(');    PRINTNUMBER(WCURSOR.X);
                            WRITECH(',');PRINTNUMBER(WCURSOR.Y);
                            WRITECH(')');WRITECH('>')
                        end;
                12: %FIND   \  FIND;
                13: % REDISPLAY   \
                       REDISPLAY
          end
      end;
    %SCHARCOMMAND   \

procedure HATCH(ANGLE:real);

      const
          HLD= 0.25; % DISTANCE BETWEEN HATCH LINES IN CMS. \
          MAXHL=100; % MAX. POSSIBLE HATCH LINES   \

      type
          PEDGE = ^LINEREC;
          LINEREC = record
                          X,DX:real;
                          DY:integer;
                          RNEXT:PEDGE
                    end;

          BUCKREC = record
                          Y:real;
                          RIGHT:PEDGE
                    end;

      var
          BUCKET: array [1..MAXHL] of BUCKREC;
          HLYDST: real; % HATCH LIN. DISTANCE IN WORLD COORD.   \
          HLBMAX: integer; % NO. OF HATCH LINES IN BUCKET   \
          SINV,COSV:real;
          AELHEAD:PEDGE; % ACTIVE EDGE LIST HEAD   \
          SENTINEL:PEDGE; % PIONTER TO LAST DUMMY RECORD
                            IN ACTIVE EDGE LIST   \
          YIX:integer; % Y INDEX OF BUCKET   \
```

```
    TEMP: PNODE;

procedure  TRANSFORM(var P:WCOORDINATE);
      var
           TEMP:WCOORDINATE;
      begin
           TEMP.X:=P.X* COSV - P.Y* SINV;
           TEMP.Y:=P.X* SINV + P.Y* COSV;
           P:=TEMP
      end;
    % TRANSFORM  \

procedure DISPLAYLINES;
      var
           TEMP: PNODE;
           SCUR,SFIX,SWCUR,SWFIX:WCOORDINATE;
      begin
           SCUR:=CURP; SFIX:=FIXP;
           SWCUR:=WCURSOR;SWFIX:=WFIX;
           TEMP:=NEWLTST;
           while TEMP <> nil do
             begin
                FIXP:=TEMP^.P0;CURP:=TEMP^.P1;
                DRAW; TEMP:=TEMP^.NEXT
             end;
      end;
    % DISPLAYLINES  \

procedure TRANSLTST;
      var
           TEMP: PNODE;
           SCUR,SFIX:WCOORDINATE;

      begin
           SCUR:=CURP; SFIX:=FIXP;
           if LISTHEAD <> nil then
             begin
                TEMP:=LISTHEAD;
                FIXP:=TEMP^.P0; CURP:=TEMP^.P1;
                TRANSFORM(FIXP);
                with MINMAXDIM ,FIXP do
                  begin
                     XMIN:=X;XMAX:=X;
                     YMIN:=Y;YMAX:=Y
                  end;
                TRANSFORM(CURP);
                UPDATEDIMENsion;
                TEMP^.P0:=FIXP; TEMP^.P1:=CURP;
                TEMP:=TEMP^.NEXT;
                while TEMP <> nil do
                  begin
                     FIXP:=TEMP^.P0; CURP:=TEMP^.P1;
                     TRANSFORM(FIXP);TRANSFORM(CURP);
                     TEMP^.P0:=FIXP;TEMP^.P1:=CURP;
```

```
                    UPDATEDIMEnsion; TEMP:=TEMP^.NEXT
                end;
            end;
        end;
      % TRANSLIST  \

procedure  INITBUCKET;
        var
            FIRSTY,TEMPY:real;
            I: integer;

        begin
            HLYDST:= SCALE * HLD;
            FIRSTY:= MINMAXDIM.YMAX + HLYDST/2;
            HLBMAX:= TRUNC((FIRSTY- MINMAXDIM.YMIN)/HLYDST)+1;
            TEMPY:= FIRSTY;
            for I:=1 to HLBMAX do
              begin
                  BUCKET[I].Y:=TEMPY;
                  BUCKET[I].RIGHT:=nil;
                  TEMPY:=TEMPY - HLYDST
              end
        end;
      % INITBUCKET  \

procedure FILLBUCKETlist;
        type
            DIRECTION =(UP,DOWN);
            PSING = ^SINGREC;
            SINGREC = record
                                  VERTEX:WCOORDINATe;
                                  TOPOLOGY: DIRECTION;
                                  PNEXT: PSING
                          end;

            LINSEG = record
                                PO,P1: WCOORDINATe
                         end;

        var
            PCUREDGE: PNODE; % POINTER TO CURRENT EDGE  \
            CUREDGE: LINSEG;
            FHLIX: integer; % FIRST HATCH LINE INTERSECTION
                                        INDEX  \
            Y:integer; CREC:LINEREC;
            E:PEDGE;
            SINGHEAD,SINGTAIL: PSING; % PTRS TO SINGULARITY LST  \
            TEMP: WCOORDINATe;

        function HORIZONTAL: boolean;
              var
                  I:integer;
              begin
                  HORIZONTAL:=false;
```

```
        I:=1;
        with CUREDGE do
          begin
              if P0.Y <> P1.Y then
                begin
                    while (BUCKET[I].Y > P0.Y) and (I<HLBMAX) do
                          I:=I+1;
                    FHLIX:=I;
                    if (BUCKET[I].Y < P0.Y) and
                    (BUCKET[I].Y < P1.Y) then
                          HORIZONTAL:=true ;
                    if I= HLBMAX then
                          if BUCKET[I].Y > P0.Y then
                                HORIZONTAL:=true;
                end
              else
                    HORIZONTAL:=true
          end;
        % WITH  \
      end;
    % HORIZONTAL  \

procedure PROCESSEDGE(var E:LINEREC; var Y:integer);
      var
        I,DELY:integer;
        DELX:real;
        SINGULAR:boolean;
        FSTY,FSTX,LSTY:real;
        P:PSING;

      procedure CHECKSINGularity;
          var
              TEMP: PSING;

          procedure  ADDSINGLIST;
                begin
                    if SINGTAIL = nil then
                      begin
                          SINGHEAD:= P;
                          SINGTAIL:=P
                      end
                    else
                      begin
                          SINGTAIL^.PNEXT:=P;
                          SINGTAIL:= P
                      end
                end;
              % ADDSINGLIST  \

          begin % CHECKSINGULARITY  \
              SINGULAR:=false;
              if SINGHEAD <> nil then
                begin
                    TEMP:=SINGHEAD;
```

```
            while (TEMP <> nil) and (not SINGULAR) do
              begin
                if TEMP^.VERTEX = P^.VERTEX  then

                      if TEMP^.TOPOLOGY <> P^.TOPOLOGY then
                            SINGULAR:=true;
                TEMP:= TEMP^.PNEXT
              end;
            % WHILE  \
            if not SINGULAR then
                ADDSINGLIST
        end
      else
            ADDSINGLIST
    end;
  % CHECKSINGULARITY  \

begin % PROCESSEDGE  \
  FSTY:=BUCKET[FHLTX].Y;
  Y:=FHLTX;
  with CUREDGE do
    begin
      FSTX:=P0.X+(P1.X-P0.X)*(FSTY-P0.Y)/(P1.Y-P0.Y);
      DELX:= (P1.X-P0.X)*HLYDST/(P0.Y-P1.Y);
      DELY:= TRUNC((FSTY-P1.Y)/HLYDST)+1;
      LSTY:=FSTY;
      for I:=1 to DELY-1 do
            LSTY:=LSTY-HLYDST;
      if (FSTY=P0.Y) or (LSTY=P1.Y) then
        begin
            if FSTY=P0.Y then
              begin
                NEW(P); P^.PNEXT:=nil;
                P^.VERTEX:=P0;
                P^.TOPOLOGY:=DOWN;
                CHECKSINGUlarity;
                if SINGULAR then
                  begin
                      FSTY:=FSTY-HLYDST;
                      FSTX:= FSTX+ DELX;
                      DELY:=DELY-1; Y:=Y+1
                  end
              end
            else
              begin
                NEW(P); P^.PNEXT:=nil;
                P^.VERTEX:=P1;
                P^.TOPOLOGY:=UP;
                CHECKSINGUlarity;
                if SINGULAR  then
                      DELY:=DELY-1
              end
        end
    end;
```

```
                %WITH  \
                E.X:=FSTX; E.DX:=DELX;
                E.DY:=DELY
            end;
          % PROCESSEDGE  \

      procedure INSERTBUCKet(E:PEDGE; Y:integer);
            begin
                E^.RNEXT:=BUCKET[Y].RIGHT;
                BUCKET[Y].RIGHT:= E
            end;

      begin % FILLBUCKETLIST  \
          SINGHEAD:=nil; SINGTAIL:=nil;
          PCUREDGE:=LISTHEAD;
          while PCUREDGE <> nil do
            begin
                CUREDGE.P0:=PCUREDGE^.P0;
                CUREDGE.P1:=PCUREDGE^.P1;
                with CUREDGE do
                      if P0.Y < P1.Y then
                        begin
                            TEMP:=P0; P0:=P1;
                            P1:=TEMP
                        end;
                if not HORIZONTAL then
                  begin
                      NEW(E);
                      PROCESSEDGe(CREC,Y);
                      E^:=CREC;
                      if E^.DY <> 0 then
                            INSERTBUCKet(E,Y)
                  end;
                PCUREDGE:= PCUREDGE^.NEXT
            end;
          % WHILE  \
      end;
    % FILLBUCKETLIST  \


procedure CHECKBUCKEt;
      var
          TEMP1,TEMP2: PEDGE;

      procedure INSERTACTIve( W:PEDGE);
            var
                W1,W2: PEDGE;
            begin
                W2:=AELHEAD;
                W1:=W2^.RNEXT;
                SENTINEL^:=W^;
                while W1^.X  < W^.X do
                  begin
                      W2:=W1; W1:=W2^.RNEXT
```

```
                end;

              N2^.RNEXT:= N;
              N^.RNEXT:= N1
            end;
          % INSERTACTIVE  \

      begin % CHECKBUCKET  \
          if BUCKET[YIX].RIGHT <> nil  then
            begin
              TEMP1:= BUCKET[YIX].RIGHT;
              while TEMP1 <> nil do
                begin
                    TEMP2:=TEMP1;
                    TEMP1:=TEMP2^.RNEXT;
                    INSERTACTIVe(TEMP2)
                end;
            end
      end;
    % CHECKBUCKET  \

procedure PSORT;
      var
          TEMP:PEDGE;
          FSTP,LSTP:PEDGE;
          NOEXCHANGE: boolean;
      begin
          if AELHEAD^.RNEXT <> SENTINEL then
            begin
              LSTP:=SENTINEL;
              repeat
                    NOEXCHANGE:=true;
                    FSTP:=AELHEAD^.RNEXT;
                    TEMP:= AELHEAD;
                    while FSTP^.RNEXT <> LSTP do
                      begin
                          if FSTP^.X > FSTP^.RNEXT^.X then
                            begin
                                TEMP^.RNEXT:= FSTP^.RNEXT;
                                FSTP^.RNEXT:= FSTP^.RNEXT^.RNEXT;
                                TEMP^.RNEXT^.RNEXT:=FSTP;
                                TEMP:=TEMP^.RNEXT;
                                NOEXCHANGE:= false
                            end
                          else
                            begin
                                TEMP:=FSTP; FSTP:=TEMP^.RNEXT
                            end
                      end;
                    % WHILE  \
                    LSTP:=FSTP
              until NOEXCHANGE
            end
      end;
```

```
    % BSORT  \

procedure UPDATELIST;
      var
          W1,W2:PEDGE;
      begin
          W2:= AELHEAD; W1:=W2^.RNEXT;
          while W1 <> SENTINEL do
             begin
                 W1^.DY:= W1^.DY-1;
                 W1^.X:= W1^.X + W1^.DX;
                 if W1^.DY=0 then
                   begin
                       W2^.RNEXT:= W1^.RNEXT;
                       W1:=W1^.RNEXT
                   end
                 else
                   begin
                       W2:=W1; W1:=W2^.RNEXT
                   end
             end;
          % WHILE  \
      end;
    % UPDATELIST  \

procedure PICKLINES;

      var
          W1:PEDGE;
          P:PNODE;
      begin
          W1:=AELHEAD^.RNEXT;
          while W1 <> SENTINEL do
             begin
                 NEW(P); P^.P0.Y:=BUCKET[YIX].Y;
                 P^.P0.X:=W1^.X; W1:=W1^.RNEXT;
                 if W1 <> SENTINEL then
                   begin
                       P^.P1.Y:=BUCKET[YIX].Y;
                       P^.P1.X:=W1^.X;
                       P^.NEXT:=NEWLIST;
                       NEWLIST:=P;
                       W1:=W1^.RNEXT
                   end

                 else    ERROR(0)
             end;

          % WHILE  \
      end;
    % PICKLINES  \

begin % HATCH  \
    NEWLIST:= nil;
```

```
        SINV:=SIND(-ANGLE); COSV:=COSD(-ANGLE);
        TRANSLIST;
        INITBUCKET;
        FILLBUCKETlist;
        NEW(AELHEAD); NEW(SENTINEL);
        AELHEAD^.RNEXT:=SENTINEL;
        for YIX:=1 to HBMAX do
          begin
              CHECKBUCKET;
              PICKLINES;
              UPDATELIST; BSORT
          end;
        SINV:=-SINV;
        TEMP:=NEWLIST;
        while TEMP <> nil do
          begin
              TRANSFORM(TEMP^.P0);
              TRANSFORM(TEMP^.P1);
              TEMP:=TEMP^.NEXT
          end;
        DISPLAYLINES

    end;
  % HATCH  \

procedure  MUCHARCOMMand;
    label
        222;
        var   P1,R2,R3:real;
        TEMP:WCOORDINATE;
        TCUR,TFIX:WCOORDINATE;
        MESSAGE:packed array[1..15] of char;
        MCH:char; AMK:integer;
        MACNAME:IDENTIFIER;

    procedure  LOADWORD;
          begin
              with CODE[ENDCODE] do
                begin
                    OPCODE:=CTP;LEV:=LEVEL;
                    D:=false; LINBEGIN:=true
                end
          end;
        % LOADWORD  \

    procedure LOCALERROR(N:integer);
          begin
              ERROR(N);
              goto 222
          end;

    procedure  LOADSTRING;
```

```
var
    I:integer;

procedure  PUTCHAR;
      begin
          if I=10 then
            begin
                GETWORD; LOADWORD; I:=0
            end;
          I:=I+1;CODE[ENDCODE].STC[I]:=CH
      end;
    %  PUTCHAR \

begin
    if (CH<> CHR(CR)) and (CH<>'$') then
      begin
          GETWORD;LOADWORD;
          CODE[ENDCODE].SI:=WCURSOR; CTP:=0;
          GETWORD;LOADWORD; I:=0;
          repeat
                PUTCHAR; GETCH
          until (CH='$')or (CH=CHR(CR));
          if CH='$' then
            begin
                PUTCHAR;GETCH
            end
          else
            begin  ENDCODE:=PC-1;
                LOCALERROR(7)
            end
      end
end;
  % LOADSTRING \

procedure  EDITOR(CTP:integer);

     type
        OPCRANGE=1..MAXOPCODE;
        OPCODESET= set of OPCRANGE;
     var
        SCUR,SFIX,SWFIX,SWCUR:WCOORDINATE;
        SAVPC:integer;
        SYMCODE,CIRCCODE,ARCCODE:OPCODESET;

     procedure  FLASHCURSOr;
           var
               P0,P1:SCOORDINATE;
           begin
               CLIPWINDOW;
               if VISIBLE then
                 begin
                     WORLDTOSCReen(WFIX,P0);
```

```
                WORLDTOSCReen(WCURSOR,P1);
                repeat
                      MOVETO(P0);WAIT(1);
                      MOVETO(P1);WAIT(1)
                until SKPINC
            end
      end;
    % FLASHCURSOR  \

procedure  LINEEDITOR;
      label
          10,20;

          var
          CH:char;   FOUND:boolean;

      procedure  SEARCHLINE;

            var
                K,D,XDIF,YDIF,XCDIF,YCDIF:real;
                DISTANCE,MAXD:real;
                ASLOPE:(LTONE,GEQONE);% LINE SLOPE IN ABS VALUE \
                WSDMAX:real; % MAX SEARCH DISTANCE IN WORLD COORD. \

            begin
                WSDMAX:=SLDMAX * SCALE;
                XDIF:= CURP.X-FIXP.X;
                YDIF:= CURP.Y-FIXP.Y;
                XCDIF:=CURP.X- WCURSOR.X;
                YCDIF:=CURP.Y- WCURSOR.Y;
                D:=ABS(XDIF)- ABS(YDIF);
                if D>0 then
                  begin
                      MAXD:=ABS(XDIF); ASLOPE:=LTONE
                  end
                else
                  begin
                      MAXD:=ABS(YDIF); ASLOPE:=GEQONE
                  end;
                if MAXD <= (WSDMAX/2) then
                  begin
                      if (ABS(XCDIF)< WSDMAX) and (ABS(YCDIF)<WSDMAX) then
                            FOUND:=true
                  end
                else
                  begin
                      if ASLOPE= LTONE then
                            K:=XCDIF/XDIF
                      else
                            K:=YCDIF/YDIF;
                      if (K>0) and (K<=1) then
                        begin
                            if ASLOPE=LTONE  then
                                  DISTANCE:=(YDIF*K)-YCDIF
```

```
                    else
                            DISTANCE:=(XDIF*K)-XCDIF;
                    if ABS(DISTANCE)< WSDMAX then
                            FOUND:=true
                end
            end
        end;
      %SEARCHLINE  \

procedure  DELETELINE;
      var
          I:integer;

      procedure  RESOLVECASe (var I:integer);
            begin
                if CODE[PC-1].LINBEGIN then
                  begin
                      if PC <> ENDCODE then
                            if (CODE[PC+1].LINBEGIN)or(CODE[PC+1].D)  then
                                  I:=1
                            else I:=2
                      else I:=1
                  end
                else
                  begin
                      if PC <> ENDCODE then
                            if (CODE[PC+1].LINBEGIN)or (CODE[PC+1].D) then
                                  I:=3
                            else I:=4
                      else I:=3
                  end
            end;
          % RESOLVECASE \

      begin % DELETELINE  \
          DELETED.DPC:=PC;
          DELETED.MODE:=LINEM;
          RESOLVECASe(I);
          case I of
                1:
                          begin
                              CODE[PC].D:=true;
                              CODE[PC-1].D:=true;
                              DELETED.CASENO:=1;
                              if PC=ENDCODE then LINEMODE:= false
                          end;
                2:
                          begin
                              CODE[PC].LINBEGIN:=true;
                              CODE[PC-1].D:=true;
                              DELETED.CASENO:=2
                          end;
                3:
```

```
                                  begin
                                      CODE[PC].D:=true;
                                      DELETED.CASEND:=3;
                                      if PC=ENDCODE then LINEMODE:=false
                                  end;
                      A:
                                  begin
                                      CODE[PC].LINBEGIN:=true;
                                      DELETED.CASEND:=4
                                  end
              end;
              %CASE   \
          end;
        %DELETELINE   \

    begin %LINEEDITOR   \
        FOUND:=false;
        PC:=STARTCODE;
        10:
        while (PC <= ENDCODE)and (not FOUND) do
              with CODE[PC] do
                begin
                    if not D then
                      begin
                          if OPCODE <= 6 then
                            begin
                                if LINBEGIN then
                                  begin FIXP:=LIN;NEXTINSTRUction
                                  end
                                else
                                  begin
                                      CURP:=LIN; SEARCHLINE;
                                      if not FOUND then
                                        begin
                                            FIXP:=CURP;NEXTINSTRUction
                                        end
                                  end
                            end
                          else NEXTINSTRUction
                      end
                    else NEXTINSTRUction
              end;
              % WITH & WHILE   \
        if FOUND then
          begin
              20:
              FLASHCURSOr; INCHRW(CH);
              case CH of
                  'D':
                              begin
                                  DELETELINE
                              end;
                  'C':
```

```
                                        begin FOUND:=false ; #CURSOR:=SWCUR;
                                            FIXP:=CURP; NEXTINSTRUCtion;goto 10
                                        end;
                                   'S':  ;
                                   others:
                                        begin ERROR(0);goto 20
                                        end
                              end;
                              * CASE   \
                          end
                        else ERROR(0)
                    end;
                  * LINEEDITOR  \

procedure DELETESYMBol;
      var
          I:integer;
      begin
          with DELETED do
            begin
                OPC:=PC;MODE:=SYMM;
                CIRCLE:=false
            end;
          if CODE[PC].OPCODE in CIRCCODE then
            begin DELETED.CIRCLE:=true;
                for I:=0 to 3 do
                      CODE[PC+I].D:=true
            end
          else
                for I:=0 to 4 do
                      CODE[PC+I].D:=true
      end;
    * DELETESYMBOL  \
procedure  SYMEDITOR;
      label
          10,20;
          var
          FOUND:boolean;
          SEARCHWINDow:BOX;
          CH:char;


      begin
          PC:=STARTCODE;FOUND:=false;
          10:
          while (PC<=ENDCODE)and (not FOUND) do
            begin
                with CODE[PC] do
                      if (not D) and (OPCODE in SYMCODE) then
                            if OPCODE in CIRCCODE then
                              begin
                                    with SEARCHWINDOW do
                                      begin
                                          XMIN:=CODE[PC+2].MINDIM.X;
```

```
                    YMIN:=CODE[PC+2].MINDIM.Y;
                    XMAX:=CODE[PC+3].MAXDIM.X;
                    YMAX:=CODE[PC+3].MAXDIM.Y
                end;
              if INSIDE(SEARCHWINDow,WCURSOR) then
                    FOUND:=true
              else
                    NEXTINSTRUction
            end
          else
            begin
              with SEARCHWINDow do
                begin
                    XMIN:=CODE[PC+3].MINDIM.X;
                    YMIN:=CODE[PC+3].MINDIM.Y;
                    XMAX:=CODE[PC+4].MAXDIM.X;
                    YMAX:=CODE[PC+4].MAXDIM.Y
                end;
              if INSIDE(SEARCHWINDow,WCURSOR) then
                    FOUND:=true
              else
                    NEXTINSTRUction
            end
        else  NEXTINSTRUction;
    end;
  %WHILE \
  if FOUND then
    begin
      FIXP:=CODE[PC].CENTRE;
      if  CODE[PC].OPCODE in CIRCCODE then
        begin
          CURP.X:=CODE[PC+1].ARC.RADIUS+FIXP.X;
          CURP.Y:=FIXP.Y
        end
      else
          CURP:=CODE[PC+2].LIN;
      20:
      FLASHCURSOr; INCHRW(CH);
      case CH of
          'D': DELETESYMBol;
          'C':
                begin  FOUND:=false;
                    WCURSOR:=SWCUR;
                    NEXTINSTRUction;goto 10
                end;
          'S': ;
          others:
                begin ERROR(0);goto 20
                end
      end;
      %CASE  \
    end
  else ERROR(0)
end;
```

```
   %LEVELEDITOR \
procedure LEVELEDITOR;

     begin
          PC:=STARTCODE;
          while PC<= ENDCODE do
            begin
                with CODE[PC] do
                       if (not D) and (LEV=DLEV) then
                        begin
                            if OPCODE in SYMCODE then
                                   DELETESYMBol
                            else
                                   D:=true
                        end;
                NEXTINSTRUction
            end;
          %WHILE \
          DELETED.MCDE:=LEVM;
          DELETED.LEVD:=DLEV
     end;
   % LEVELEDITOR  \
begin % EDITOR  \
   SYMCODE:=[7..15];CIRCCODE:=[7..9];
   ARCCODE:=[10..15];
   SAVPC:=PC;SCUR:=CURP;SFIX:=FIXP;
   SWFIX:=WFIX;SWCUR:=WCURSOR;
   case CTP of
          20: LINEEDITOR;
          21: SYMEDITOR;
          23: LEVELEDITOr
   end;
   PC:=SAVPC;CURP:=SCUR;FIXP:=SFIX;
   WFIX:=SWFIX;WCURSOR:=SWCUR; MOVE
end;
% EDITOR  \


procedure DISPLAYLEvel(L:integer);
     var
          SCUR,SFIX,SWCUR,SWFIX:WCOORDINATE;
          SPC:integer;
     begin
          SCUR:=CURP;SFIX:=FIXP;SWFIX:=WFIX;
          SWCUR:=WCURSOR;PC:=STARTCODE;
          LEVDISPLAY:=true; DLEV:=L;
          INITSCREEN;
          INTERPRET; LEVDISPLAY:=false;
          CURP:=SCUR;FIXP:=SFIX;WFIX:=SWFIX;
          WCURSOR:=SWCUR;MOVE
     end;
   % DISPLAYLEVEL  \

procedure INITHATCH(L:integer);
```

```
    var
        SCUR,SFIX,SWCUR,SWFIX:WCOORDINATE;
    begin
        SCUR:=CURP;SFIX:=FIXP;
        SWCUR:=WCURSOR; SWFIX:=WFIX;
        LISTHEAD:=nil; LISTTAIL:=nil;
        LEVDISPLAY:=true; DLEV:=L;
        HATCHFLAG:=true;PC:=STARTCODE;
        INTERPRET; HATCHFLAG:=false;
        LEVDISPLAY:=false;
        CURP:=SCUR;FIXP:=SFIX;
        WCURSOR:=SWCUR; WFIX:=SWFIX
    end;
  % INITHATCH  \

procedure  ACCEPTPOLYgon;
    var
        SCCCODE:set of 1..NSCC;
        CMD:char;
        FIRSTP,ENDPOLY:boolean;
        P0,P1:WCOORDINATE;
    begin
        ENDPOLY:=false; FIRSTP:=true;
        SCCCODE:=[1..8,12];
        LISTHEAD:=nil;LISTTATL:=nil;
        repeat
              INCHRW(CMD); while CMD=' ' do INCHRW(CMD);
              CTP:=NSCC;SCCOMTAB[0]:=CMD;
              while  CMD <> SCCOMTAB[CTP] do
                      CTP:=CTP-1;
              if CTP in SCCCODE then
                      SCHARCOMMAnd
              else
              case  CMD of
                      'X':
                                 begin
                                     SCHARCOMMAnd;
                                     if FIRSTP then
                                       begin
                                           P0:=WCURSOR;P1:=P0;
                                           FIRSTP:=false
                                       end
                                     else
                                       begin
                                           P0:=P1;P1:=WCURSOR;
                                           ENTERLIST(P0,P1)
                                       end
                                 end;

                      'E':  ENDPOLY:=true;

                      others: ERROR(0)
              end;
        until  ENDPOLY
```

```
    end;
  % ACCEPTPOLYGON \

procedure  SENDMESSAGe;
    var
        I:integer;
    begin
        for I:=1 to 15 do
              WRITECH(MESSAGE[I]);
        OUTCHR(BEL)
    end;
  % SENDMESSAGE  \

procedure  LOADHATCHLine;
    var
        TEMP:PNODE;
    begin
        TEMP:=NEWLIST; CTP:=1;
        while  TEMP <> nil do
          begin
              GETWORD;LOADWORD;
              CODE[ENDCODE].LIN:=TEMP^.P0;
              GETWORD; LOADWORD;
              with CODE[ENDCODE] do
                begin
                    LINBEGIN:=false; LIN:=TEMP^.P1
                end;
              TEMP:=TEMP^.NEXT
          end;
        % WHILE  \
        PC:=ENDCODE+1
    end;
  % LOADHATCHLINE  \


procedure MACROHANDLer(MNAME:IDENTIFIER;SCALE,ROTD:real);

    var
        P0,P1,OFFSET,ORIGIN:WCOORDINATe;
        SINV,COSV:real;
        SAVPC,DLEN,SP,EP:integer;

    procedure  MACTRANSFOrm(var T:WCOORDINATe);
          var
              P:WCOORDINATe;
          begin
              T.X:=T.X-ORIGIN.X;
              T.Y:=T.Y-ORIGIN.Y;
              P.X:=(T.X*COSV-T.Y*SINV)*SCALE+ OFFSET.X;
              P.Y:=(T.X*SINV+T.Y*COSV)*SCALE+ OFFSET.Y;
              T:=P
          end;
        % MACTRANSFORM \
```

```
procedure  SETMINMAXDim(var P0,P1:WCOORDINATe);
    var
        TEMP,SCUR,SFIX:WCOORDINATe;
        I:integer;
        PARRAY: array[1..4] of WCOORDINATe;

    begin
        PARRAY[1]:=P0;PARRAY[3]:=P1;
        PARRAY[2].X:=P1.X;PARRAY[2].Y:=P0.Y;
        PARRAY[4].X:=P0.X;PARRAY[4].Y:=P1.Y;
        TEMP:=PARRAY[1];MACTRANSFOrm(TEMP);
        with MINMAXDIM ,TEMP do
          begin
              XMIN:=X;XMAX:=X;
              YMIN:=Y;YMAX:=Y
          end;
        SCUR:=CURP; SFIX:=FIXP;
        FIXP:=PARRAY[1]; CURP:=PARRAY[2];
        for I:=1 to 2 do
          begin
              MACTRANSFOrm(FIXP);
              MACTRANSFOrm(CURP);
              UPDATEDIMEnsion;

              FIXP:=PARRAY[3];CURP:=PARRAY[4]
          end;
        CURP:=SCUR; FIXP:=SFIX;
        with MINMAXDIM do
          begin
              P0.X:=XMIN;P0.Y:=YMIN;
              P1.X:=XMAX;P1.Y:=YMAX
          end;
    end;
  % SETMINMAXDIM  \

begin % MACROHANDLER \
    RESET(MAINF,FNMAC);
    if EOF(MAINF)  then
      begin
        MESSAGE:=' MAC FILE EMPTY';
        SENDMESSAGe; goto 222
      end;
    DLEN:=MAINF^.FDIR.DSIZE;
    SEARCHDIREctory(MAINF,MNAME);
    if not DIRFOUND then
      begin
        MESSAGE:=' MACRO NOTFOUND';
        SENDMESSAGe; goto 222
      end;
    with MAINF^.FDIR do
      begin
        SP:=STRTP+DLEN+1;
        EP:=ENDP+DLEN+1
```

```
  end;
RESET(MAINF,FNMAC);
LOADMEMORY(MAINF,SP,EP);
SINV:=SIND(ROTD); COSV:=COSD(ROTD);
OFFSET:=WCURSOR; ORIGIN:=CODE[PC+2].LIN;
SAVPC:=PC;
while PC<= ENDCODE do
  begin
    with CODE[PC] do
      begin
        case OPCODE of
              1,2,3,4,5,6:
                    begin
                      LEV:=LEVEL; MACTRANSFOrm(LIN)
                    end;
              7,8,9:
                    begin
                      LEV:=LEVEL;
                      P0:=CODE[PC+2].MINDIM;
                      P1:=CODE[PC+3].MAXDIM;
                      SETMINMAXDim(P0,P1);
                      CODE[PC+2].MINDIM:=P0;
                      CODE[PC+3].MAXDIM:=P1;
                      MACTRANSFOrm(CENTRE);
                      with CODE[PC+1].ARC do
                           RADIUS:=RADIUS*SCALE;
                    end;
              10,11,12,13,14,15:
                    begin
                      LEV:=LEVEL;
                      P0:=CODE[PC+3].MINDIM;
                      P1:=CODE[PC+4].MAXDIM;
                      SETMINMAXDim(P0,P1);
                      CODE[PC+3].MINDIM:=P0;
                      CODE[PC+4].MAXDIM:=P1;
                      MACTRANSFOrm(CENTRE);
                      with CODE[PC+1].ARC do
                           RADIUS:=RADIUS*SCALE;
                    end;
              31:
                    begin
                      LEV:=LEVEL;
                      MACTRANSFOrm(ST)
                    end
        end;
        % CASE \
      end;
    % WITH  \
    NEXTINSTRUction
  end;
% WHILE \
with MINMAXDIM ,OFFSET do
  begin
    XMIN:=X;YMIN:=Y; XMAX:=X;YMAX:=Y
```

```
            end;
         LOADDIMENSion:=true; PC:=SAVPC;
         INTERPRET; LOADDIMENSion:=false;
         with MINMAXDIM do
           begin
               with CODE[SAVPC].MINDIM do
                 begin
                     X:=XMIN; Y:=YMIN
                 end;
               with CODE[SAVPC+1].MAXDIM do
                 begin
                     X:=XMAX; Y:=YMAX
                 end;
           end;
         GETWORD; with CODE[ENDCODE] do
           begin
               OPCODE:=61; LEV:=LEVEL
           end;
         PC:=PC+1
      end;
    & MACROHANDLER \

begin &MUCHARCOMMAND  \
    GETSYM;
    if SY=IDEN then
      begin
         CTP:=NMCC; MCCOMTAB[0]:=ID;
         while  MCCOMTAB[CTP] <> ID do CTP:=CTP-1
      end
    else
         LOCALERROR(8);
    case CTP of
         0: ERROR(1); &COMMAND ERROR \
         1,2,3,4,5,6: & LINES  \
                   begin   GETSYM;
                       if SY=CRTN  then
                         begin
                             TFTX:=WFTX;TCUR:=WCURSOR
                         end
                       else
                             if SY=IDEN  then
                               begin
                                   if ID<>'L     ' then
                                          LOCALERROR(9);
                                   GETNUMBER(R1);
                                   if not NUMFLAG then
                                          LOCALERROR(4);
                                   if R1<0 then LOCALERROR(2);
                                   GETSYM;R2:=0;
                                   if SY<>CRTN then
                                          if SY=IDEN then
                                            begin
                                                if ID<>'A     ' then
                                                       LOCALERROR(10);
```

```
                                                GETNUMBER(R2);
                                                if not NUMFLAG then
                                                     LOCALERROR(4);
                                            end
                                           else
                                                LOCALERROR(10);
                                  TFIX:=WCURSUR;TCUR.X:=TFIX.X+R1*COSD(R2);
                                  TCUR.Y:=TFIX.Y+R1*SIND(R2)
                             end
                            else
                                  LOCALERROR(9);
                       if LINEMODE  then
                         begin
                            if TFIX <> CURP then
                              begin
                                  GETWORD; LOADWORD;
                                  CODE[ENDCODE].LIN:=IFIX
                              end
                         end
                       else
                         begin LINEMODE:=true;
                              GETWORD; LOADWORD;
                              CODE[ENDCODE].LIN:=IFIX
                         end;
                       GETWORD; LOADWORD;
                       with CODE[ENDCODE]  do
                         begin
                              LINBEGIN:=false; LIN:=TCUR
                         end;
                       %WITH \
                       INTERPRET; GETSYM
                   end;
                 % LINES  \
         16,17: % POSITION CURSOR  \
                   begin   GETSYM;
                       if SY=CRTN then
                              LOCALERROR(11);
                       R1:=0; R2:=0;
                       repeat
                              if SY=IDEN then
                                begin
                                    if ID='X    ' then GETNUMBER(R1)
                                    else
                                          if ID='Y    ' then GETNUMBER(R2)
                                          else
                                                LOCALERROR(11);
                                    if not NUMFLAG then
                                          LOCALERROR(4);

                                end
                              else
                                    LOCALERROR(11);
                              GETSYM
                       until SY=CRTN;
```

```
            TEMP.X:=R1;TEMP.Y:=R2;
            if CTP=17 then
              begin
                  TEMP.X:=TEMP.X+WCURSOR.X;
                  TEMP.Y:=TEMP.Y+WCURSOR.Y
              end;
            if INSIDE(WINDOW,TEMP) then
              begin
                  WCURSOR:=TEMP; MOVE
              end
            else ERROR(12)
        end;
      % POSITIONCURSOR \
18: % SET LARGE CURSOR STEP  \
        begin
            GETNUMBER(R1);
            if not NUMFLAG then
                  LOCALERROR(4);
            LCURSTEP:= R1;  GETSYM
        end;
      % LARGE CURSOR STEP  \

27: % MOVE CURSOR BY SPECIFIED LEN & ANGLE \
        begin
            GETSYM;
            if SY=IDEN  then
              begin
                  if ID<>'L      ' then
                        LOCALERROR(9);
                  GETNUMBER(R1);
                  if not NUMFLAG then
                        LOCALERROR(4);
                  if R1<0 then LOCALERROR(2);
                  GETSYM;R2:=0;
                  if SY<>CRTN then
                        if SY=IDEN then
                          begin
                              if ID<>'A      ' then
                                    LOCALERROR(10);
                              GETNUMBER(R2);
                              if not NUMFLAG then
                                    LOCALERROR(4)
                          end
                        else
                              LOCALERROR(8);
                  TCUR.X:=WCURSOR.X+R1*COSD(R2);
                  TCUR.Y:=WCURSOR.Y+R1*SIND(R2);
                  if INSIDE(WINDOW,TCUR) then
                    begin WCURSOR:=TCUR; MOVE
                    end
                  else ERROR(12)
              end
            else
                  LOCALERROR(9);
```

```
      end;
    % MOVE CURSOR \

10: % SET LEVEL  \
      begin
        GETSYM;
        if SY=INTCON then
          begin
            if (INUM<0)or(INUM>MAXLEVEL) then LOCALERROR(6);
            LEVEL:=INUM ;LINEMODE:=false
          end
        else ERROR(5);
        GETSYM
      end;
    % SET LEVEL  \

7,8,9: %CIRCLES  \
      begin    GETSYM;
        if SY in [CRTN,IDEN] then
           if SY=IDEN  then
             begin
               if ID<>'R      ' then
                     LOCALERROR(13);
               GETNUMBER(R1);
               if not NUMFLAG then
                     LOCALERROR(4);
               if R1<0 then LOCALERROR(3)
             end
           else
             begin   R1:=(WFIX.X-WCURSOR.X);
               R2:=(WFIX.Y-WCURSOR.Y); R1:=SQRT(R1*R1+R2*R2)
             end
        else
           LOCALERROR(8);
        if LINEMODE then
           LINEMODE:= false;
        GETWORD;LOADWORD;
        CODE[ENDCODE].CENTRE:=WCURSOR;
        GETWORD;LOADWORD;
        CODE[ENDCODE].ARC.RADIUS:=R1;
        GETWORD;LOADWORD;
        with CODE[ENDCODE].MINDIM do
          begin
            X:=WCURSOR.X-R1;
            Y:=WCURSOR.Y-R1
          end;
        GETWORD;LOADWORD;
        with CODE[ENDCODE].MAXDIM  do
          begin
            X:=WCURSOR.X+R1;
            Y:=WCURSOR.Y+R1
          end;
        INTERPRET; GETSYM
      end;
```

```
          % CIRCLES   \
10,11,12,13,14,15:% ARCS   \
            begin
                GETSYM;
                if SY=IDEN  then
                  begin
                      if ID<> 'A       ' then
                            LOCALERROR(10);
                      GETNUMBER(R1);
                      if not NUMFLAG then
                            LOCALERROR(4);
                      R2:= WFIX.X-WCURSOR.X;

                      R3:=WFIX.Y-WCURSOR.Y;
                      R2:=SQRT(R2*R2 + R3*R3)
                  end
                else
                      LOCALERROR(10);
                if LINEMODE then
                      LINEMODE:= false;
                GETWORD; LOADWORD;
                CODE[ENDCODE].CENTRE:=WCURSOR;
                GETWORD;LOADWORD;
                with CODE[ENDCODE].ARC do
                  begin
                      RADIUS:=R2; ANGLE:=R1
                  end;
                GETWORD;LOADWORD;
                CODE[ENDCODE].LIN:=WFIX; %STARTING POINT OF ARC  \
                GETWORD;LOADWORD;
                %--------------------------------------- \
                % THIS IS TO MAKE SURE THAT INTERPRETER DRAWS THE SYMBOL  \
                with CODE[ENDCODE].MINDIM do
                  begin
                      X:= WINDOW.XMIN;
                      Y:= WINDOW.YMIN
                  end;
                GETWORD;LOADWORD;
                with CODE[ENDCODE].MAXDIM do
                  begin
                      X:= WINDOW.XMAX;
                      Y:= WINDOW.YMAX
                  end;
                %--------------------------------------- \
                with MINMAXDIM ,WFIX  do
                  begin
                      XMIN:=X;YMIN:=Y;
                      XMAX:=X; YMAX:=Y
                  end;
                LOADDIMENSion:=true;
                INTERPRET;
                LOADDIMENSion:=false;
                with MINMAXDIM do
                  begin
```

```
                      with CODE[PC-1].MAXDIM do
                        begin
                            X:=XMAX; Y:=YMAX
                        end;
                      with CODE[PC-2].MINDIM do
                        begin
                            X:=XMIN; Y:=YMIN
                        end
                  end;
                % WITH MINMAXDIM  \
                GETSYM
            end;
          % ARCS   \
20,21,34,35: %SEARCH LINE,SYMBOL  ,MACRO,STRING& EDIT  \
          EDITOR(CTP);
22: % SHOW LEVEL  \
            begin
                GETSYM;
                if SY=INTCON then
                  begin
                      if (INUM<0) or (INUM>MAXLEVEL) then
                            LOCALERROR(6);
                      DISPLAYLEVel(INUM)
                  end
                else
                      LOCALERROR(5)
            end;

23: % DELETE LEVEL  \
            begin
                GETSYM;
                if SY=INTCON then
                  begin
                      if (INUM<0) or (INUM>MAXLEVEL) then LOCALERROR(6);
                      DLEV:=INUM;EDITOR(CTP)
                  end
                else
                      LOCALERROR(5)
            end;
          % DELETE LEVEL  \

24: % END AND SAVE  \
            begin   GARCOLLECT; SAVEPIC:=true; DRAFTOVER:=true
            end;
25: % END AND NO SAVE  \
            begin  SAVEPIC:=false; DRAFTOVER:=true
            end;

26: % HATCH  \
            begin
                MARK(AMK);
                GETSYM;
                if (SY=INTCON) and ((INUM< MAXLEVEL)and(INUM>=0)) then
                      INITHATCH(INUM)
```

```
            else
                if (SY=IDEN)and (ID='POLY ') then
                  begin
                      MESSAGE:=' MARK  POLYGON ';
                      SENDMESSAGe;
                      ACCEPTPOLYgon
                  end
                else
                      LOCALERROR(8);
            HATCH(HANGLE);
            MESSAGE:='OK? TYPE Y/N:  ';
            SENDMESSAGe; INCHRV(MCH);
            if MCH= 'Y' then  LOADHATCHLine
                ;
            RELEASE(AMK);  GETSYM
        end;
      % HATCH LINE  \

28: % SETWINDOW \
        begin
            R1:=WCURSOR.X-WFIX.X;
            R2:=WCURSOR.Y-WFIX.Y;
            if R1>=R2 then
                  SETWINDOW(WFIX.X,WCURSOR.X,WFIX.Y,WFIX.Y+R1)
            else
                  SETWINDOW(WFIX.X,WFIX.X+R2,WFIX.Y,WCURSOR.Y);
            SETCURSORStep; INITSCREEN;
            PC:=STARTCODE;
            INTERPRET
        end;
      % SETWINDOW \

29:   %  DEFAULT WINDOW \
        begin
            SETWINDOW(0,SCALE*24.4,0,SCALE*24.4);
            SETCURSORStep; PC:=STARTCODE;
            INITSCREEN; INTERPRET
        end;
      %  DEFAULT WINDOW \

30:        % CALL MACRO  \

        begin
            if DWGSTATUS.DTYP<> MAIN  then
              begin
                  MESSAGE:=' NO MACRO CALL ';
                  SENDMESSAGe; goto 222
              end;
            GETSYM;
            if SY<> IDEN then LOCALERROR(8);
            MACNAME:=ID; GETSYM; R1:=1;R2:=0;
            if SY<> CRTN then
            repeat
                  if SY= IDEN then
```

```
                              begin
                                  if ID='S     ' then GETNUMBER(R1)
                                  else
                                        if ID='R     ' then GETNUMBER(R2)
                                        else
                                              LOCALERROR(14);
                                  if not NUMFLAG then  LOCALERROR(4)
                              end
                            else
                                  LOCALERROR(14);
                            GETSYM
                      until  SY= CRTN;
                      MACROHANDLer(MACNAME,R1,R2)
                  end;
                  % CALL MACRO \

            31: % STRING \
                  begin
                      GETSYM;
                      if SY= DOLLAR  then
                        begin  LOADSTRING  ;
                            INTERPRET;
                            if LINEMODE then LINEMODE:=false
                        end
                      else
                            LOCALERROR(7);
                      GETSYM
                  end;
                  %STRING \

            32:  %   SET HATCH ANGLE \
                  begin
                      GETNUMBER(R1);
                      if not NUMFLAG then
                            LOCALERROR(4);
                      HANGLE:=R1; GETSYM
                  end;

            33: % SET MACRO ORIGIN \
                  MACORIGIN:=NCURSOR;

            others: ERROR(0)
      end;
      %CASE  \
      222:
  end;
begin %DRIVER  \
   repeat

      CC:=0;INCHRW(CH);while CH=' ' do INCHRW(CH);
      SCCOMTAB[0]:=CH;
      CTP:=NSCC;
```

```
                while CH <>SCCOMTAB[CTP] do CTP:=CTP-1;
                if CTP <>0 then  SCHARCOMMAnd
                else
                  begin
                      if (CH = CHR(CR)) then
                        begin INCHRW(CH);
                            ERROR(0); goto 111
                        end;
                      FILLBUFFER(BUFFUL);
                      if  BUFFUL then
                        begin
                            GETCH; %ONE CHAR LOOK AHEAD  \
                            FUCHARCOMMand;
                            if CH <> CHR(CR) then ERROR(0);
                        end;
                      WRITECH('>')
                  end;
                %ELSE  \
                111:
            until DRAFTOVER;
        end;
    %DRIVER  \

procedure GETFILENAMes;
      var
          FBUFF:FILENAME;
          CH:char;

      procedure GETNAME;
            label
                11;
                var
                I,J:integer;
                CH:char;
            begin
                11:
                for I:=1 to 9 do
                      FBUFF[I]:=' ';
                WRITE(TTY,'FILE:');BREAK;
                INCHRW(CH);
                while CH = ' ' do INCHRW(CH);
                if CH=CHR(CR) then
                  begin
                      INCHRW(CH); WRITELN(TTY);
                      goto 11
                  end;
                I:=0;
                while CH in DIGITS+ALPHAS+['.'] do
                  begin
                      if CH='.' then I:=6
                      else
                            if I<9 then
                              begin
                                  I:=I+1; FBUFF[I]:=CH
```

```
                    end;
                INCHRW(CH)
            end;

          if CH <> CHR(CR) then
            begin
                while CH<>CHR(CR) do INCHRW(CH);
                INCHRW(CH); WRITELN(TTY);

                WRITELN(TTY,'ILLEGAL CHAR IN FILE NAME');
                goto 11
            end
          else  INCHRW(CH)
        end;
      %GETNAME \

    begin  % GETFILENAMES  \
      MACFILFLAG:=false;
      WRITELN(TTY,'DO YOU HAVE ANY MACRO FILE?');
      WRITE(TTY,'TYPE  Y/N :'); BREAK;INCHRW(CH);
      if CH='Y' then
        begin
          WRITELN(TTY); GETNAME;
          MACFILFLAG:=true;
          FNMAC:=FBUFF;FBUFF[7]:='O';
          FNMACBAK:=FBUFF
        end;
      WRITELN(TTY);
      WRITELN(TTY,'GIVE FILE NAME IN WHICH DRAWING TO BE STORED');
      GETNAME; FNMAIN:=FBUFF;
      FBUFF[7]:='O'; FNMBAK:=FBUFF
    end;
   % GETFILENAMES  \

procedure SEARCHDIRECtory;
    var
      I,DLEN:integer;
    begin
      DIRFOUND:=false;
      I:=F^.FDIR.DSIZE; DLEN:=I;
      while (I<>0)and (not DIRFOUND) do
        begin
          I:=I-1;GET(F);
          if F^.FDIR.NAME=ID then
            begin
                DIRFOUND:=true;
                DIX:=DLEN-I
            end
        end
    end;
   % SEARCHDIRECTORY  \
procedure LOADMEMORY;
    var
      I:integer;
```

```
      begin
          I:=1;
          while I<SP do
            begin
                GET(F); I:=I+1
            end;
          for I:=SP to EP do
            begin
                ENDCODE:=ENDCODE+1;
                CODE[ENDCODE]:=F^.FCODE;
                GET(F)
            end
      end;
    % LOADMEMORY  \

procedure MONITOR;
      label
          333;
          var
          CTP,T:integer;
          KJNG,BUFFUL:boolean;
          NLEN,SP,EP:integer;
          FNAME, BAKFNAME:FILENAME;
          P:real; DIREMPTY:boolean;


      procedure COPYFILE(var F1,F2:FILETYP);
            begin
                while not EOF(F1) do
                  begin
                      F2^:=F1^;
                      PUT(F2); GET(F1)
                  end
            end;
          % COPYFILE  \
      procedure PRINTDIRECtory(var F:FILETYP);
            var
                I,J:integer;
            begin
                J:=F^.FDIR.DSIZE;
                for I:=1 to J do
                  begin
                      GET(F);WRITELN(TTY,F^.FDIR.NAME);
          /           BREAK
                  end
            end;
          % printdirectory  \

      procedure INSERTNEWPic(var F1,F2:FILETYP);
            var
                SZ,I,J:integer;
                OFFSET:integer;
            begin
                if not DIREMPTY then
```

```
  begin
      SZ:=F1^.FDIR.DSIZE;J:=SZ;
      SZ:=SZ+1;F1^.FDIR.DSIZE:=SZ;
      F2^.FDIR:=F1^.FDIR;PUT(F2);
      for I:=1 to J do
        begin
            GET(F1);F2^.FDIR:=F1^.FDIR;
            PUT(F2)
        end;
      OFFSET:=F1^.FDIR.ENDP;
      with F2^.FDIR do
        begin
            NAME:=DWGSTATUS.DNAME;
            STRTP:=OFFSET+1;
            ENDP:=OFFSET+ENDCODE
        end;
      PUT(F2);
      for I:=1 to OFFSET do
        begin
            GET(F1);F2^.FCODE:=F1^.FCODE;
            PUT(F2)
        end
  end
 else
  begin
      F2^.FDIR.DSIZE:=1; PUT(F2);
      with F2^.FDIR do
        begin
            NAME:=DWGSTATUS.DNAME;
            STRTP:=1;ENDP:=ENDCODE
        end;
      PUT(F2)
  end;
 for I:=STARTCODE to ENDCODE do
  begin
      F2^.FCODE:=CODE[I]; PUT(F2)
  end
end;
% insertnewpic  \

procedure  REPLACEOLDpic(var F1,F2:FILETYP);
   var
       I,OLDDIF,NEWDIF,DIF:integer;
   begin
       I:=DIX;
       while I <> 0 do
         begin
             F2^:=F1^;PUT(F2);GET(F1); I:=I-1
         end;
       NEWDIF:=ENDCODE-STARTCODE+1;
       with F1^.FDIR do  OLDDIF:=ENDP-STRTP+1;
       DIF:=NEWDIF-OLDDIF;
       F2^:=F1^;F2^.FDIR.ENDP:=F2^.FDIR.ENDP+DIF;
       PUT(F2);GET(F1); I:=DLEN-DIX;
```

```
        while I <> 0 do
          begin
            F2^:=F1^; with F2^.FDIR do
              begin
                STRTP:=STRTP+DIF;ENDP:=ENDP+DIF
              end;
            PUT(F2); GET(F1); I:=I-1
          end;
        % WHILE \
        I:=SP-(DLEN+1);
        while I<> 1 do
          begin
            F2^:=F1^; PUT(F2); GET(F1);I:=I-1
          end;
        for I:=SP to EP do GET(F1);
        for I:=STARTCODE to ENDCODE do
          begin
            F2^.FCODE:=CODE[I]; PUT(F2)
          end;
        while not EOF(F1) do
          begin
            F2^:=F1^; PUT(F2); GET(F1)
          end
      end;
    % REPLACEOLDPIC \

procedure  DELETEPIC(var  F1,F2:FILETYP);
      var
        I,DIFF,SZ: integer;
      begin
        F2^.FDIR.DSIZE:=DLEN-1;
        PUT(F2); GET(F1);
        for I:= 1 to DIX - 1 do
          begin
            F2^:= F1^; PUT(F2); GET(F1)
          end;
        with F1^.FDIR do DIFF:=ENDP-STRTP+1;
        GET(F1);  I:=DLEN-DIX;
        while I<> 0 do
          begin
            F2^:= F1^;
            with  F2^.FDIR  do
              begin
                STRTP:=STRTP -DIFF;
                ENDP:= ENDP -DIFF
              end;
            PUT(F2); GET(F1); I:=I -1
          end;
        I:= SP -(DLEN+1);
        while I<> 1 do
          begin
            F2^:= F1^; PUT(F2); GET(F1); I:= I-1
          end;
```

```
            for I:= SP to EP do  GET(F1);
            while not EOF(F1) do
              begin
                F2^:= F1^; PUT(F2); GET(F1)
              end
         end;
       % DELDESCRIPTC  \

procedure INITMACHEAD;
         var
            I:integer;
         begin
            for I:=1 to 3 do
              begin
                GETWORD; with CODE[ENDCODE] do
                  begin
                      OPCODE:=60; LEV:=LEVEL; D:=false;LINBEGIN:=true
                  end
              end
         end;
       % INITMACHEAD \

procedure LOCALERROR(N:integer);
         begin
            ERROR(N);
            goto 333
         end;

begin % monitor  \
       NOECHO; SETWINDOW(0,SCALE*24,4,0,SCALE*24,4);
       MONITORMODe:=true; SETCURSORStep;
       CLEARSCREEn; WRITELN(TTY);BREAK;
       KJOB:=false;
       repeat
            WRITECH('$');
            CC:=0;INCHRW(CH);
            if CH = CHR(CR) then
              begin
                INCHRW(CH);LOCALERROR(0);
              end;
            FILLBUFFER(BUFFUL);
            if not BUFFUL then goto 333;
            GETCH;% one char look ahead  \
            GETSYM;
            if SY= IDEN then
              begin
                CTP:=NMONC;MONCOMTAB[0]:=ID;
                while MONCOMTAB[CTP] <> ID do
                      CTP:=CTP-1;
              end
            else
                LOCALERROR(8);
            case  CTP of
```

```
0:ERROR(1);
1,2:
      begin % type commands \
          GETSYM;
          if SY <> IDEN then
                 LOCALERROR(8);
          if CTP= 1 then FNAME:=FNMAIN
          else
                 FNAME:= FNMAC;
          RESET(MAINF,FNAME);
          if EOF(MAINF) then
            begin
                 WRITELN(TTY);
                 WRITE(TTY,ID,'  not found, directory empty');
                 BREAK
            end
          else
            begin
                 DLEN:=MAINF^.FDIR.DSIZE;
                 SEARCHDIREctory(MAINF,ID);
                 if DIRFOUND  then
                   begin
                       with MAINF^.FDIR do
                         begin
                             SP:=STRTP+DLEN+1;
                             EP:=ENDP+DLEN+1
                         end;
                       RESET(MAINF,FNAME);
                       STARTCODE:=1;ENDCODE:=0;
                       LOADMEMORY(MAINF,SP,EP);
                       CLEARSCREEn; PC:=STARTCODE;
                       INTERPRET
                   end
                 else
                   begin
                       ERROR(0); WRITELN(TTY);
                       WRITE(TTY,ID,'  not found');BREAK
                   end
            end
      end;
    %type statement \

3:
      begin % set scale  \
          GETNUMBER(R);
          if not NUMFLAG then
                 LOCALERROR(4);
          SCALE:=R;
          SETWINDOW(0,SCALE*24.4,0,SCALE*24.4);
          SETCURSORStep
      end;
    % set scale  \

4,5:
```

```
        begin %print directory  \
            if CTP=4 then FNAME:=FNMAIN
            else
                  FNAME:=FNMAC;
            RESET(MAINF,FNAME);WRITELN(TTY);
            if EOF(MAINF) then
              begin
                  WRITE(TTY,'directory empty');
                  BREAK
              end
            else
                  PRINTDIRECtory(MAINF)
        end;
      % print directory  \
6,7:
        begin % draft   ,MACRO \
            GETSYM;
            if SY <> IDEN then
                  LOCALERROR(8);
            if CTP=6 then
              begin
                  FNAME:=FNMAIN; BAKFNAME:=FNMBAK
              end
            else
              begin
                  if not MACFILFLAG then
                    begin  WRITELN(TTY);
                          WRITE(TTY,'MACRO LIB FILE NOT SPECIFIED ');BREAK;
                          LOCALERROR(0)
                    end;
                  FNAME:=FNMAC; BAKFNAME:=FNMACBAK
              end;

            with DWGSTATUS do
              begin
                  DNAME:=ID;
                  if CTP=6 then  DTYP:=MAIN
                  else DTYP:=MACRO;
                  STATUS:=NEW1
              end;
            RESET(MAINF,FNAME);
            if not EOF(MAINF) then
              begin
                  DLEN:=MAINF^.FDIR.DSIZE;
                  DIREMPTY:=false;
                  SEARCHDIRECtory(MAINF,ID);
                  if DIRFOUND then
                    begin
                        DWGSTATUS.STATUS:=OLD;
                        with MAINF^.FDIR do
                          begin
                              SP:=STRTP+DLEN+1;
                              EP:=ENDP+DLEN+1
```

```
                end;
              INITIALISE;SCREENHEAD:='edit:       ';
              INITSCREEN;
              RESET(MAINF,FNAME);
              LOADMEMORY(MAINF,SP,EP);
              PC:=STARTCODE;INTERPRET
            end
          else
            begin
              INITIALISE;
              SCREENHEAD:='DRAW:      ';
              INITSCREEN;
              if CTP=7 then INITMACHEAD;
            end
        end
      else
        begin
          DIREMPTY:=true;
          SCREENHEAD:='draw:      ';
          INITIALISE; INITSCREEN;
          if CTP=7 then INITMACHEAD;
        end;
      DRAFTOVER:=false;
      MONITORMODe:=false;
      WRITECH('>'); DRIVER;
      MONITORMODe:=true;
      if SAVEPIC then
        begin
          if CTP=7 then CODE[3].LIN:=MACORIGIN;
          if DWGSTATUS.STATUS = NEW1 then
            begin
              RESET(MAINF,FNAME);
              REWRITE(MAINBAK,BAKFNAME);
              COPYFILE(MAINF,MAINBAK);
              REWRITE(MAINF,FNAME);
              RESET(MAINBAK,BAKFNAME);
              INSERTNEWPic(MAINBAK,MAINF);
              RESET(MAINF,FNAME)
            end
          else
            begin
              RESET(MAINF,FNAME);
              REWRITE(MAINBAK,BAKFNAME);
              REPLACEOLDpic(MAINF,MAINBAK);
              RESET(MAINBAK,BAKFNAME);
              REWRITE(MAINF,FNAME);
              COPYFILE(MAINBAK,MAINF);
              RESET(MAINF,FNAME)

            end;
        end;
      CLEARSCREEN
```

```
      end;
    % draft   , MACRO\

",9:  begin % delete  \
          GETSYM;
          if ST<> IDEN then
                LOCALERROR(8);
          if CTP= 8 then
            begin
                FNAME:=FNMAIN; BAKFNAME:=FNMBAK
            end
          else
            begin
                FNAME:=FNMAC; BAKFNAME:=FNMACBAK
            end;
          RESET(MAINF,FNAME);
          if EOF(MAINF)  then
            begin
                WRITELN(TTY);
                WRITE(TTY,ID,'  NOT FOUND. DIRECTORY EMPTY');
                BREAK
            end
          else
            begin
                DLEN:=MAINF^.FDIR.DSIZE;
                SEARCHDIREctory(MAINF,ID);
                if DIRFOUND then
                      if  DLEN=1  then
                        begin
                            REWRITE(MAINF,FNAME);
                            RESET(MAINF,FNAME)
                        end

                      else
                        begin

                            with MAINF^.FDIR  do
                              begin
                                  SP:=STRTP+DLEN+1;
                                  EP:=ENDP+DLEN+1
                              end;
                            REWRITE(MAINBAK,BAKFNAME);
                            RESET(MAINF,FNAME);
                            DELETEPIC(MAINF,MAINBAK);
                            RESET(MAINBAK,BAKFNAME);
                            REWRITE(MAINF,FNAME);
                            COPYFILE(MAINBAK,MAINF);
                            RESET(MAINF,FNAME)
                        end
                else
                  begin
                      ERROR(0); WRITELN(TTY);
```

```
                                        WRITE(TTY,TD,'   NOT FOUND');
                                        BREAK
                                    end
                                end
                          end;
                          % DELETE  \
                    10: % kjob  \
                          KJOB:=true
               end;
               % case  \

               333:
               WRITELN(TTY);BREAK;
            until KJOB
         end;
         % monitor  \
      begin % main  \
         CLEARSCREEN;
         INITIALISE;
         GETFILENAMES;
         MONITOR
      end.
```

```
                                                  WRITE(TTY,TU,' NOT FOUND');
                                                  BREAK
                                              end
                                        end
                                  end;
                               % DELETE  \
                         13: % KJOB  \
                               KJOB:=true
                  end;
                  % CASE  \
                  333:
                  WRITELN(TTY);BREAK;
            until KJOB
       end;
     % monitor  \
begin % main  \
     CLEARSCREEN;
     INITIALISE;
     GETFILENAMES;
     MONITOR
end.
```